

Databases

**[http://www.dcs.fmph.uniba.sk/~plachetk
/TEACHING/RLDB2014](http://www.dcs.fmph.uniba.sk/~plachetk/TEACHING/RLDB2014)**

<http://www.dcs.fmph.uniba.sk/~sturc/databazy/rldb>

Tomáš Plachetka, Ján Šturc

**Faculty of mathematics, physics and informatics,
Comenius University, Bratislava**

Summer 2013–2014

Course organisation: lecture, tutorials

Lectures&tutorials: Thursday 14:00–18:00, F109

Consultations: standard hours are Wednesday 14:00–16:00.

Consultations at other times are a matter of agreement (suggest time via e-mail)

No consultations during examination period!

Course organisation: evaluation

Evaluation criteria: Tutorials are graded, 40% are necessary minimum to qualify for the final exam. To pass the course, necessary minimum is 40% from the final exam, and 50% average from both tutorials and the exam (equally weighted)

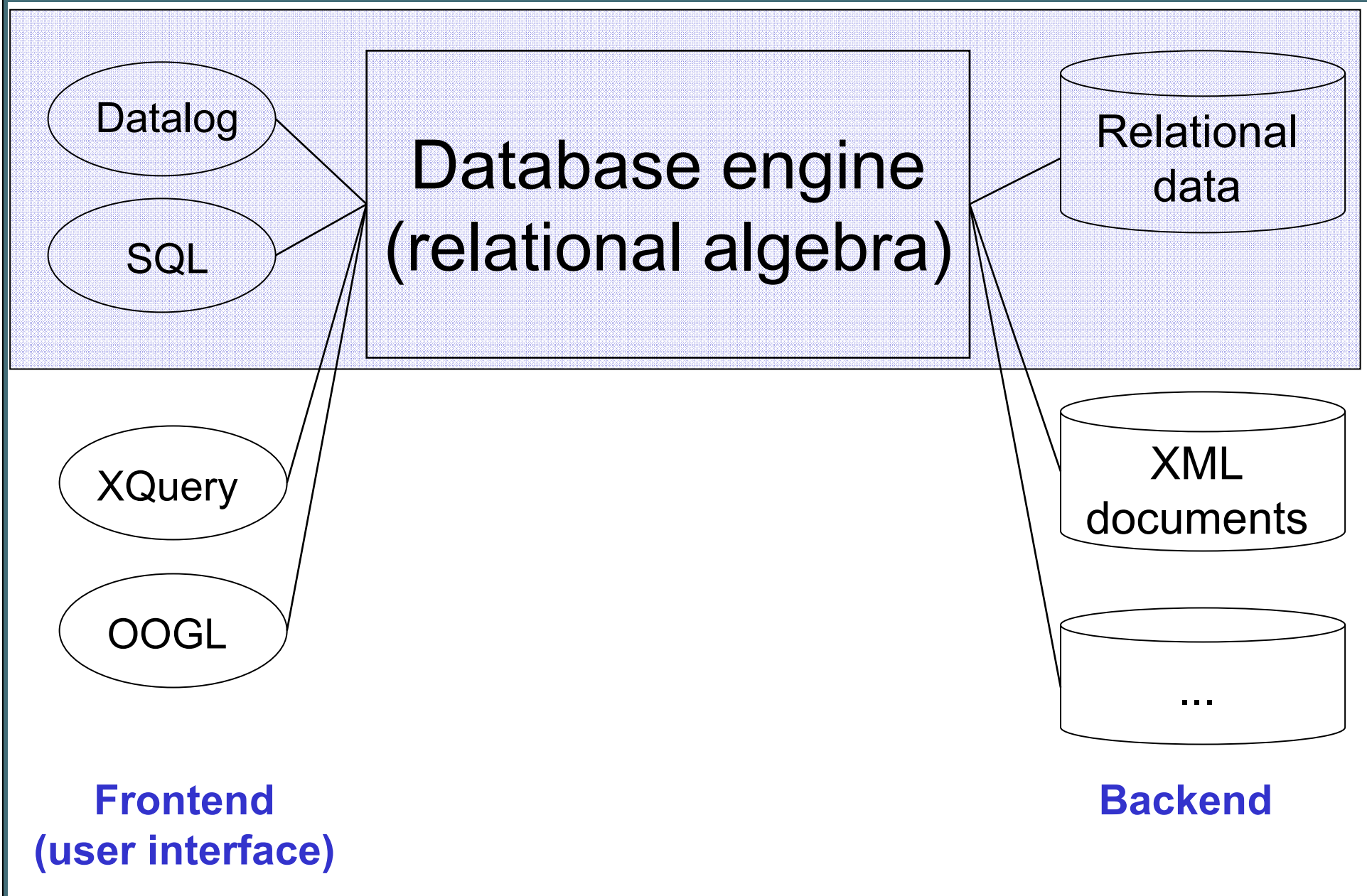
Recommended reading

- H. Garcia-Molina, J.D. Ullman, J. Widom: Database Systems, The Complete Book, Prentice Hall 2003
- S. Abiteboul, R. Hull, V. Vianu: Foundations of Databases, Addison-Wesley, 1995
- C. Zaniolo: Advanced Database Systems, Morgan Kaufmann, 1997
- S. W. Dietrich, S. D. Urban: An Advanced Course in Database Systems (Beyond Relational Databases), Pearson Prentice Hall, 2005
- P.A. Bernstein, V. Hadzilacos, N. Goodman: Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1987
- ...

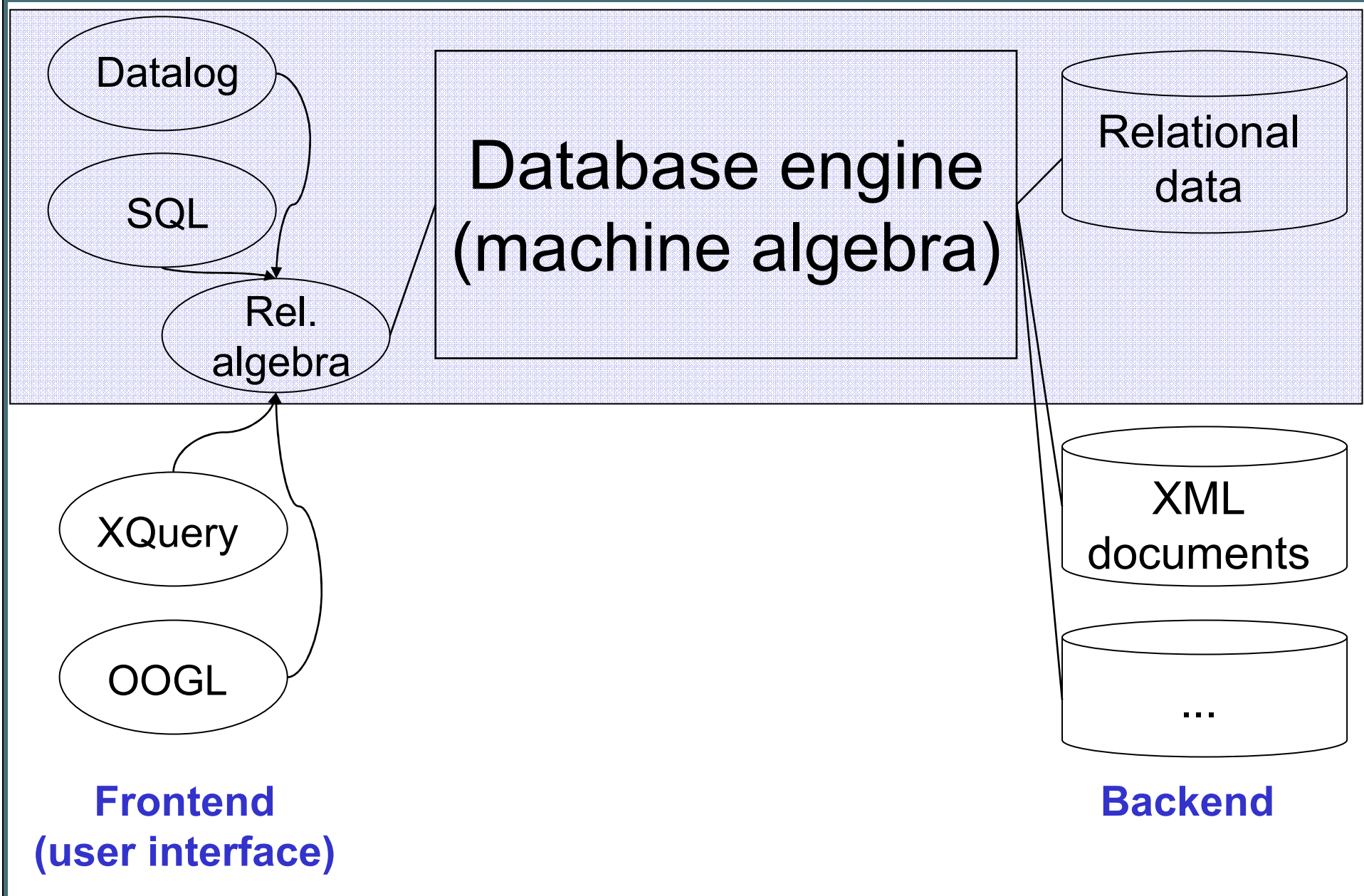
Implementation and optimisation of database systems with focus on relational (and deductive) databases

- Presently, other kinds of databases are also used and studied: network, object, document, ... However, they have no underlying mathematical theory so far and are a challenge for research. (Some trends may be just programmers' escape from mathematics.)
- **Graph is a binary relation** $[N, E \subseteq N \times N]$. Tree is a special graph. Replacement of relations with trees is therefore only a matter of language used to talk about structures

Structure of a modern database system



Structure of a modern database system



Subproblems

- **Semantics of frontend programming languages**, models of programs
- Adding functional symbols to Datalog (and other languages)
- **Computational models**: (semi-) naive evaluation, Pure Prolog
- Translation of Datalog and SQL to relational algebra (and *shredding*, mapping of XML documents to relational schemas)
- **Optimisation of queries**:
 - at the level of Datalog
 - at the level of relational algebra (minimisation of intermediate relations)
 - at the level of physical algebra (minimisation of execution time, minimisation of number of disk operations)

Functional symbols (functors) in Datalog

- Functions return **values of any type**. A function has a name and arity (number of arguments). Functions of arity 0 are constants
- Mathematically, function can be replaced with characteristic predicate “which stores the graph of the function”:

$$f(X_0, \dots, X_{n-1}) = y \Leftrightarrow p_f(X_0, \dots, X_{n-1}, Y)$$

Functional dependency $X_0, \dots, X_{n-1} \rightarrow Y$ holds

- We will usually use **functional symbols to represent structured data**. In some programming languages (Prolog, CL, ...), they are used to **encode numbers** (natural, rational, ...)

Functional symbols: arithmetics on natural numbers

Arithmetics on natural numbers:

$\text{nat}(0).$

$\text{nat}(s(X)) \leftarrow \text{nat}(X).$

$\text{add}(X, 0, X) \leftarrow \text{nat}(X).$

$\text{add}(X, s(Y), s(Z)) \leftarrow \text{nat}(X), \text{nat}(Y), \text{nat}(Z), \text{add}(X, Y, Z).$

$\text{le}(X, Y) \leftarrow \text{nat}(X), \text{nat}(Y), \text{add}(X, _, Y). \quad /* \text{ less equal } */$

$\text{mult}(X, 0, 0) \leftarrow \text{nat}(X).$

$\text{mult}(X, s(Y), Z) \leftarrow \text{nat}(X), \text{nat}(Y), \text{nat}(Z), \text{mult}(X, Y, V), \text{add}(V, X, Z).$

$\text{mult}(X, Y, Z) \leftarrow \text{nat}(X), \text{nat}(Y), \text{nat}(Z), \text{mult}(Y, X, Z).$

Functional symbols: arithmetics on natural numbers

$\text{nat}(0)$. $\text{nat}(s(X)) \leftarrow \text{nat}(X)$. **Unary encoding of natural numbers.**

Natural numbers are represented as terms: 0 , $s(0)$, $s(s(0))$, ...

$\text{nat}(X)$ means that X is a natural number.

$\text{add}(X, Y, Z) \equiv Z = X + Y$

$\text{mult}(X, Y, Z) \equiv Z = X \times Y$

$\text{le}(X, Y) \equiv X \leq Y$

We will sometimes rely on built-in arithmetic predicates instead of using the definitions above (built-in predicates are presumably more efficient). However, when details are important, we will define numbers from scratch

Functional symbols: arithmetics

Homework (related more to computability theory than databases):

Implement all elementary mathematical operations (+, -, \times , /) and functions (sqrt, sin, cos, log, exp) on natural numbers.

Propose a representation of whole and rational numbers

Terms as data structures

Fragment of an XML document:

```
<addr>  
  <place>  
    <street>Baker_Street</street>  
    <nr>221B</nr>  
  </place>  
  <city>London  
    <pcode>NW1_6XE</pcode>  
  </city>  
</addr>
```

a(p(s(Baker_Street), nr(221B)), c(London, pc(NW1_6XE)))

Terms as data structures

Binary tree:

- Predicates:

tree(T) true when T is binary tree

label(L) true when L is a label of a node

- Functional symbols:

null empty tree

node(L, T1, T2) tree with a root L,
 left subtree T1, right subtree T2

- Rules:

tree(null).

tree(node(L, T1, T2)) ← label(L), tree(T1), tree(T2).

Computation of Datalog programs with functors

- Naïve and semi-naïve evaluation works, but is not necessarily finite. E.g. query ?- nat(X) never finishes.
- Least fixpoint exists, but is not necessarily finite. (E.g. predicate nat „comprises“ all natural numbers, predicate tree „comprises“ all binary trees etc.)

Definition: Herbrand universe for a program P is a set of all terms which arise during an evaluation of P

Terms

Definition:

- Variable is a **term**.
- Let t_0, t_{n-1} be terms. Let f be an n -ary functional symbol. Then $f(t_0, \dots, t_{n-1})$ is a **term**.

Any constant is also a term (a function with no arguments).

Informally, term is any text which can be correctly used as an argument of a predicate:

- constants
- variables
- functors

Substitutions

Definition: **Substitution** is function $\tau: V \rightarrow T$, where V is a set of variables and T is a set of terms. A substitution is applied to terms.

Substitutions are denoted as $\tau = [\{X_i \mapsto t_i\}_{i=1, \dots, n}]$

The result of an application of substitution φ on term t is term $s = t \tau$, which is obtained by replacement of variables common in both t and τ by terms determined by τ . The replacement is made with **all variables (“in parallel”, i.e. **once for each variable**)**

Ordering and equivalence of terms

Definition: A **term t is at least as general as a term s** (denoted $t \supseteq s$), if a substitution τ exists such that $s = t \tau$.

Definition: **Terms t a s are equivalent** (denoted $t \cong s$), if $t \supseteq s$ and $s \supseteq t$

It holds: Two terms are equivalent, $t \cong s$, if they differ only in naming of variables

This structure is called **term algebra**, although it resembles rather a lattice. Maxima are variables, minima are constant terms (ground terms, which do not contain variables)

Composition of substitutions

Definition: **Substitution σ is a composition of substitutions ρ and τ** (denoted $\sigma = \rho \circ \tau$), if for each term t holds $t\sigma = (t\tau)\rho$

Although substitutions are functions, composition of substitutions is not composition of functions (they are partial functions)

Definition: **Substitution τ is at least as general as substitution σ** (denoted $\tau \supseteq \sigma$), if an substitution ρ exists such that $\sigma = \tau \circ \rho$

Definition: **Substitutions τ and σ are equivalent** (denoted $\tau \cong \sigma$), if $\tau \supseteq \sigma$ and $\sigma \supseteq \tau$

Composition of substitutions

Empty substitution, partial identical substitution and all substitutions which only permute variables' names are equivalent

Substitution which is **not equivalent to identical substitution** is called **specialisation**

Term matching

Term matching is a specialised unification task (we will deal with unification later): Consider a term t and a ground term s (i.e. term s does not contain variables). The task is to find whether $t \supseteq s$; and if so, then **find substitution τ such that $s = t \tau$**

Solution: recursive descent (into the structure of the terms). Begin with empty substitution τ and run the following **procedure $\text{match}(s, t)$** . When it returns TRUE, then term t is at least as general as term s and τ is the substitution which proves it (we say that term t matches term s). When the procedure returns FALSE, then no solution exists (we say that term t does not match term s)

Term matching

```
boolean match(t, s) {  
    if (t is a variable) {  
        if ( $\tau(t)$  is undefined) {  
             $\tau(t) = s$ ;  
            return TRUE;  
        }  
        else  
            return  $\tau(t) == s$ ;  
    }  
    else if ( $t == f(t_0, \dots, t_{k-1})$  and  $s == f(s_0, \dots, s_{k-1})$ ) {  
        for ( $i = 0$ ;  $i < k$ ;  $i++$ ) {  
            if (! match( $t_i, s_i$ ))  
                return FALSE;  
        }  
        return TRUE;  
    }  
    else  
        return FALSE;  
}
```

Unification: variants of tasks

Unification solves equations in term algebra. There are several variants:

1. Let t and s be terms. Find most general substitutions τ and σ such that $t \tau = s \sigma$
2. Let t and s be terms. Find most general substitution σ such that $t \sigma = s \sigma$. (This is a classical mathematical task on solving equations.)
3. Let t and s be terms. Find most general substitutions τ and σ such that $\rho \sigma = s \sigma$. This variant is called weak unification.
4. If terms are bound by a more specialised algebra than equality of terms (on the level of texts), then we talk about paramodulation. („Modulo“ is meant with respect to the set of the other equations.)

Unification: variants of tasks

We will only consider variant 2:

Let t and s be terms. Find most general substitution σ such that
 $t \sigma = s \sigma$

As to the other variants:

- The most general solution of variant 3 is obtained so that ρ renames the variables in t in such a way that they are distinct from variables in s . We denote $t' = t \rho$ and solve variant 2: $t' \tau = s \sigma$
- Solution of variant 1 can be obtained from the solution of variant 3 by setting $\tau = \rho \circ \sigma$
- Solution to variant 1 with the use of the previous two statements is the most general solution to variant 1

Unification: algorithms

Two main approaches to unification:

- Manipulation of the set of equations: J.Herbrand (1930), A.Martelli, U.Montanari (1982)
- Manipulation of trees or dags which represent terms: J.A.Robinson (1965), J.Corbin, M.Bidoit (1983), I.Prívara, P.Ružička (1989)

(This is only a selection of representative algorithms.)

Unification: Herbrand (1930)

Let E be a system of equations. Let σ be initially an empty substitution.

```
while ( $E \neq \emptyset$ ) {  
    select an equation  $e \in E$ ;  
     $E := E - \{e\}$ ;  
    if ( $e$  is  $X = t$  or  $e$  is  $t = X$ ) {  
        if ( $X$  occurs in  $t$ )  
            return solution does not exist;  
        else {  
             $\sigma = \sigma \circ [x \mapsto t]$ ;  
             $E = E[x \mapsto t]$ ;  
        }  
    }  
    else  
        if ( $e$  is of the form  $f(s_0, \dots, s_{n-1}) = f(t_0, \dots, t_{n-1})$ )  
             $E = E \cup \{s_0 = t_0, \dots, s_{n-1} = t_{n-1}\}$ ;  
        else return(solution does not exist)  
}
```

Unification: Ullman (Huet?)

Both the terms are represented with trees or (better) dags

1. Define equivalence of nodes as follows:

- Roots of the terms are equivalent
- If two nodes are equivalent, then their sons are also equivalent (in order)
- Leafs denoted by the same variable or same constant are equivalent

2. Compute symmetric, reflective and transitive closure from these equivalence classes

3. Assign substitution to a class of equivalence:

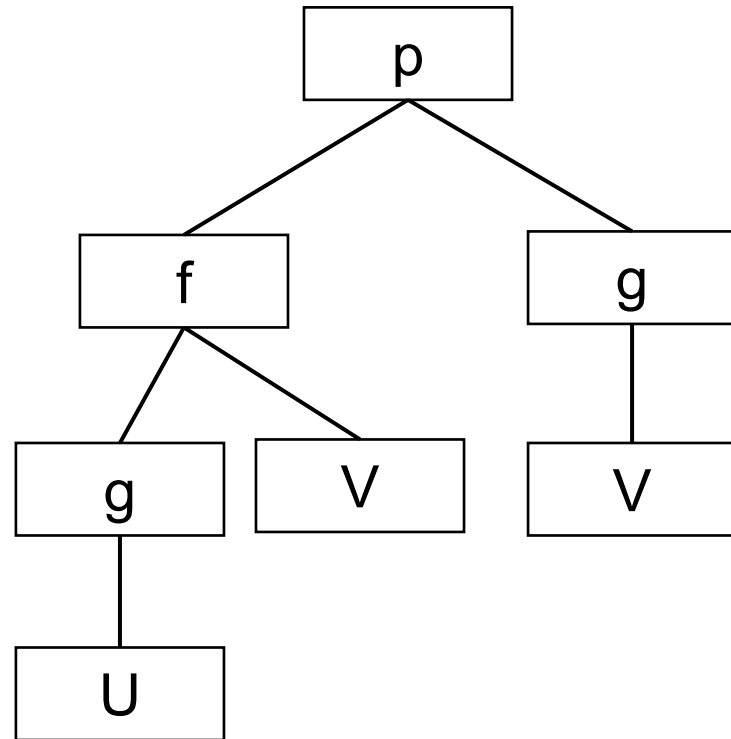
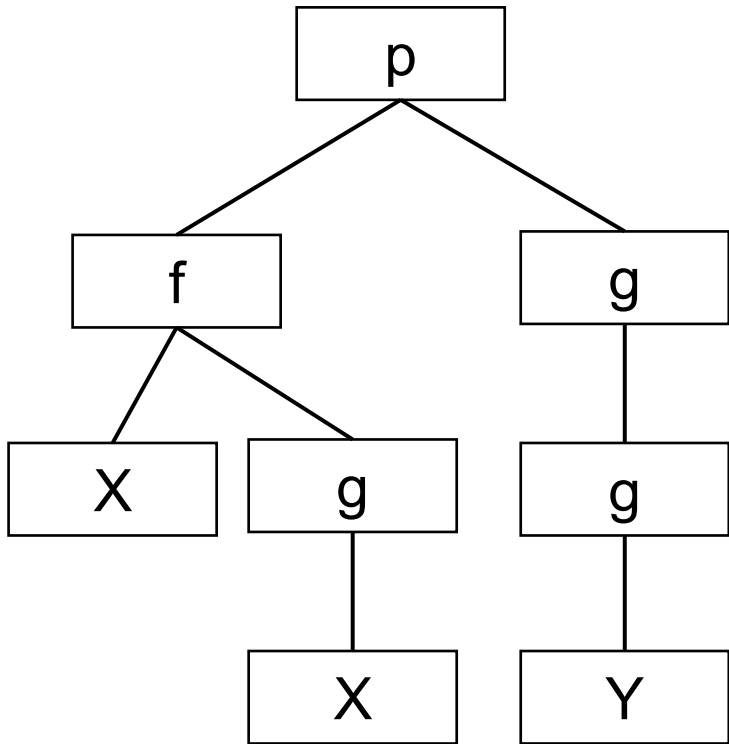
- If a class contains two nodes with different functional symbols, then solution does not exist
- If a class contains only variables, then choose one and map the others to the chosen one
- If a class contains variables as well as functors, then map all variables to the node with the functor

4. Apply occurs-check

Unification: Ullman (Huet?)

Example:

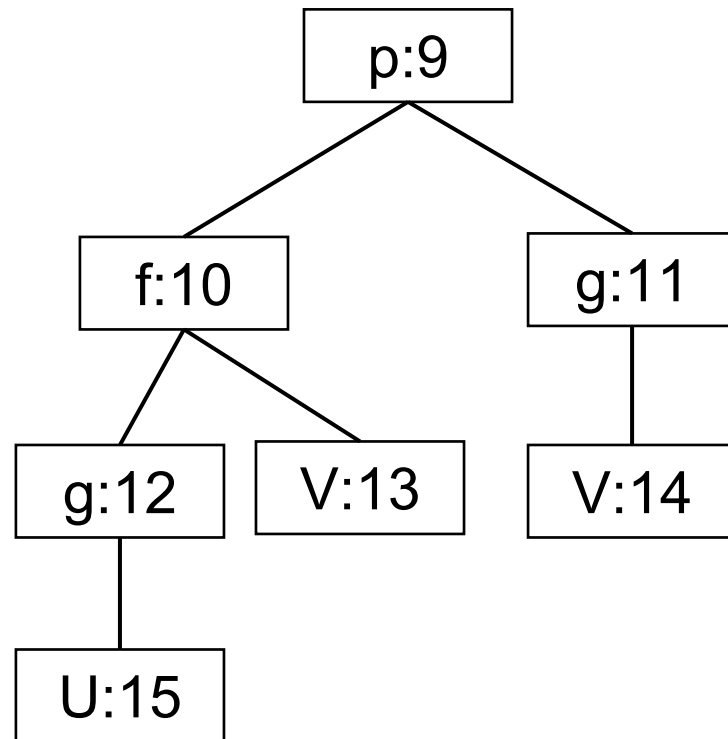
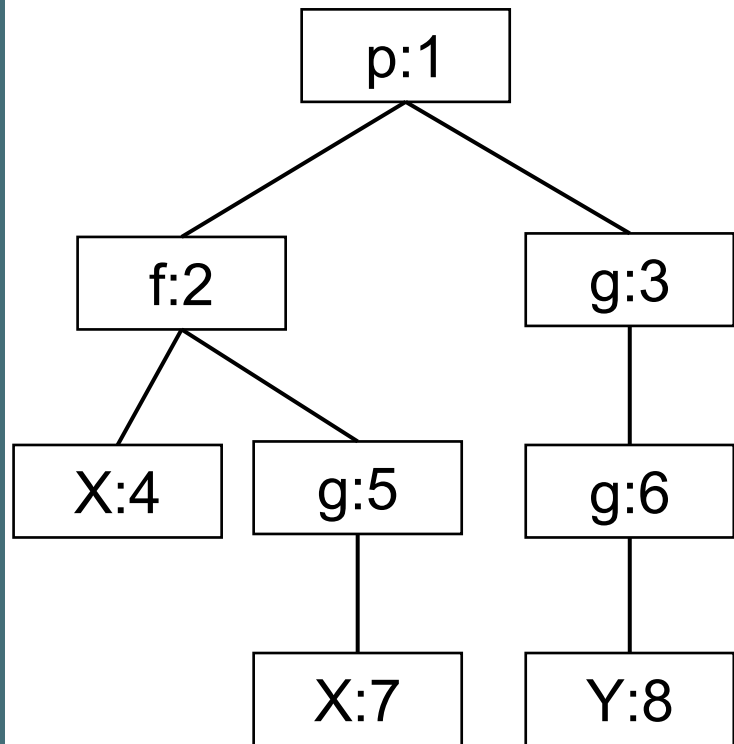
$$p(f(X, g(X)), g(g(Y))) = p(f(g(U), V), g(V))$$



Unification: Ullman (Huet?)

Example:

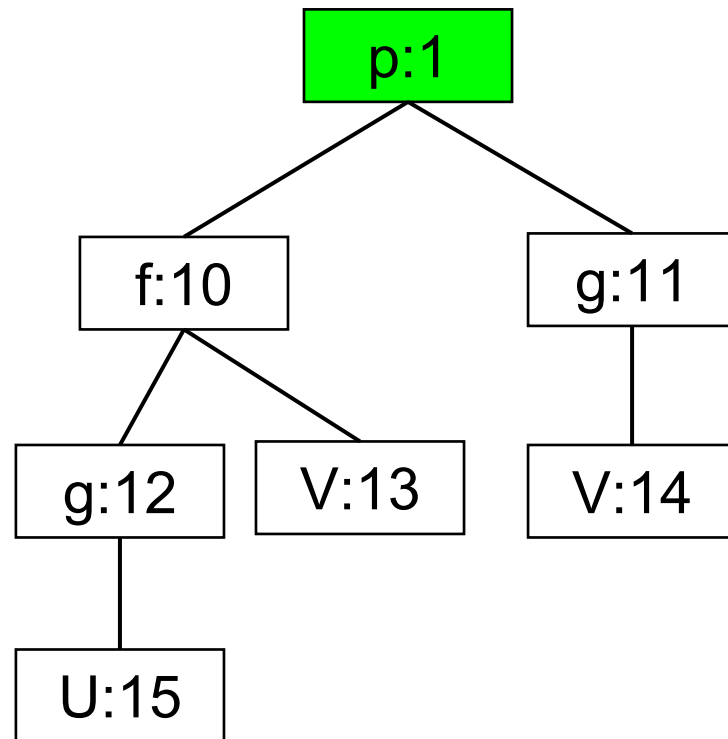
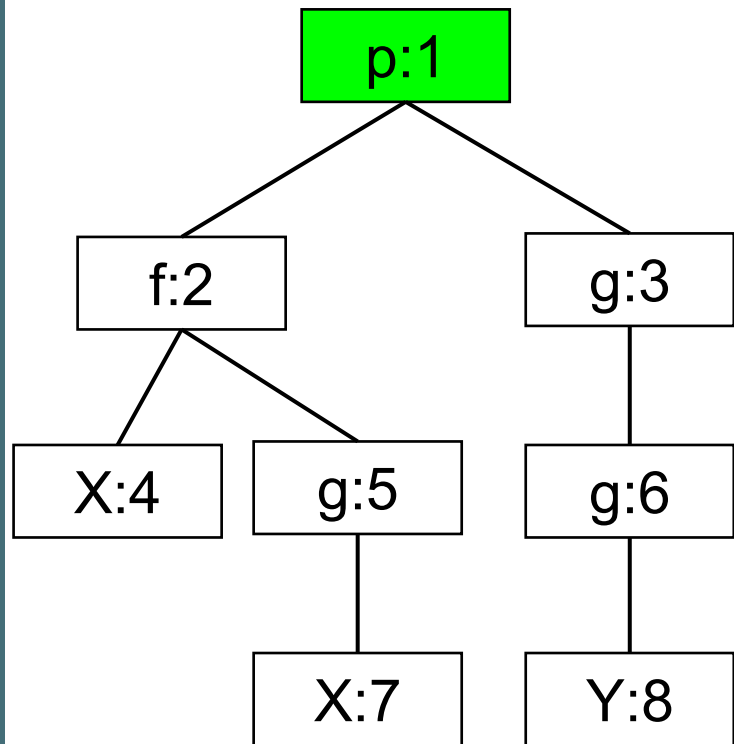
$$p(f(X, g(X)), g(g(Y))) = p(f(g(U), V), g(V))$$



Unification: Ullman (Huet?)

Example:

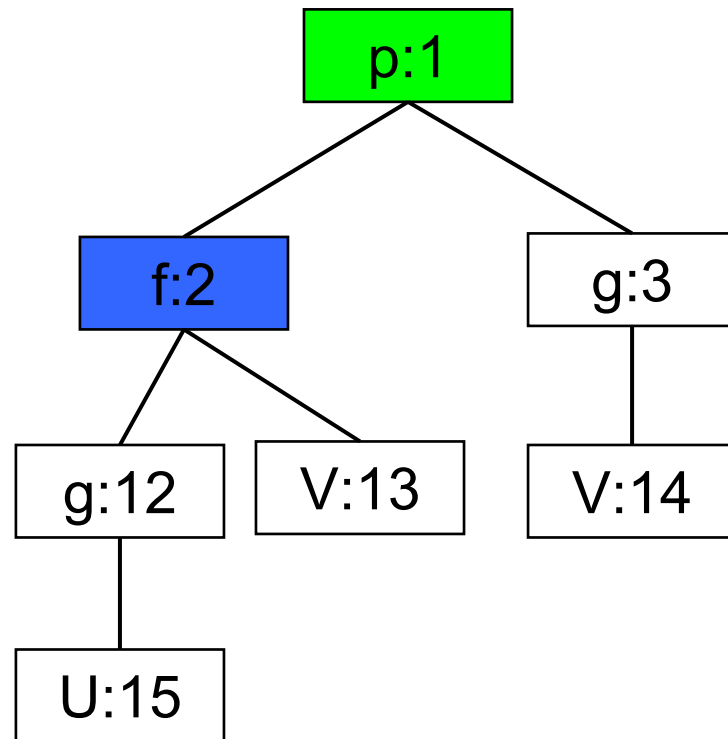
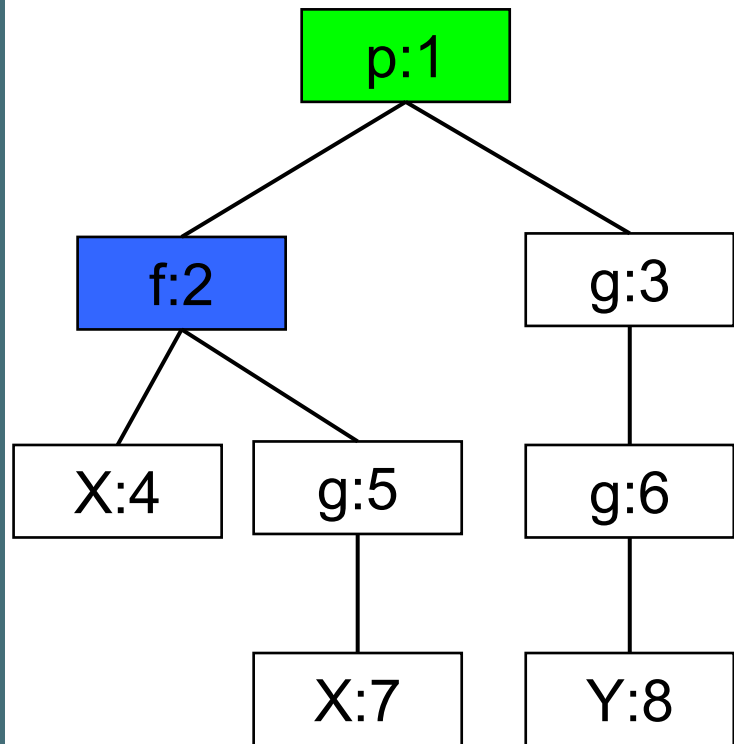
$$p(f(X, g(X)), g(g(Y))) = p(f(g(U), V), g(V))$$



Unification: Ullman (Huet?)

Example:

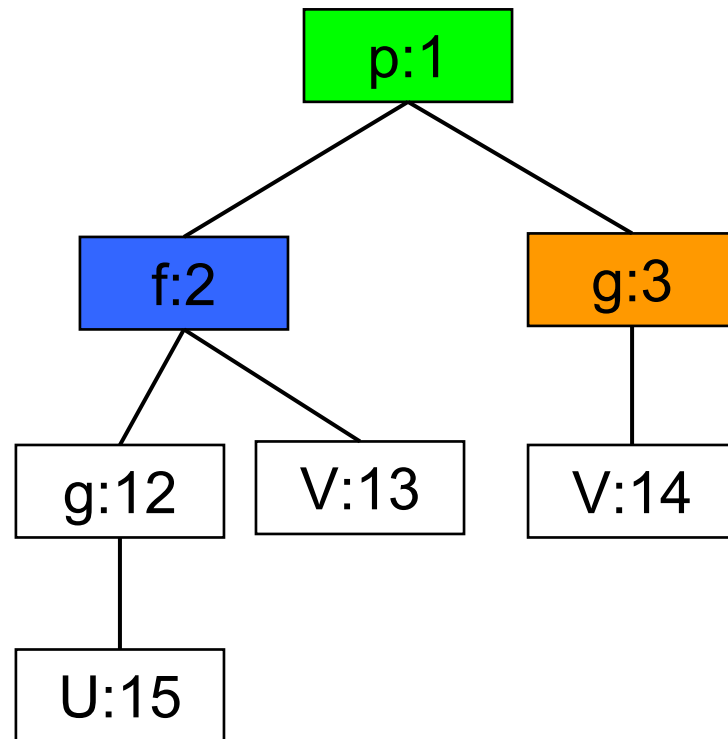
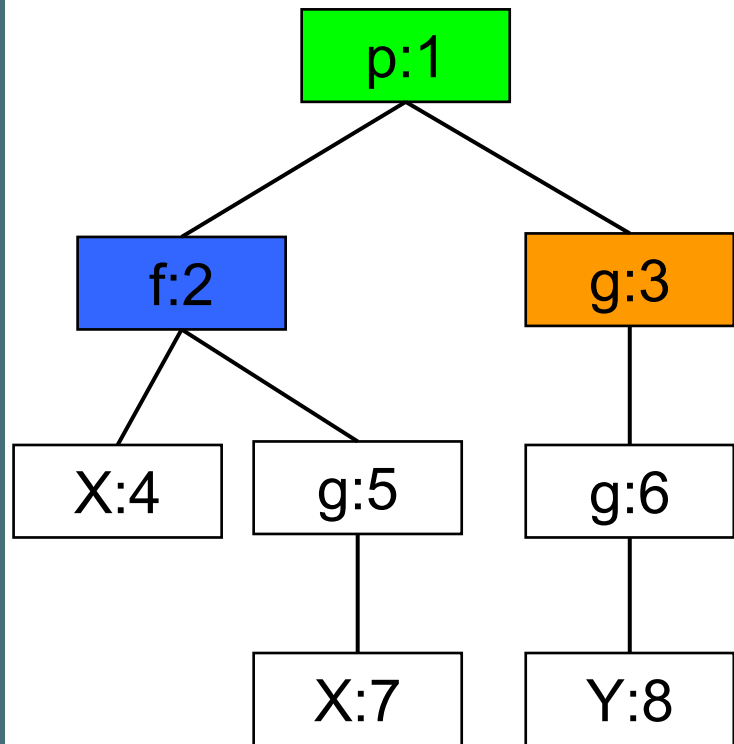
$$p(f(X, g(X)), g(g(Y))) = p(f(g(U), V), g(V))$$



Unification: Ullman (Huet?)

Example:

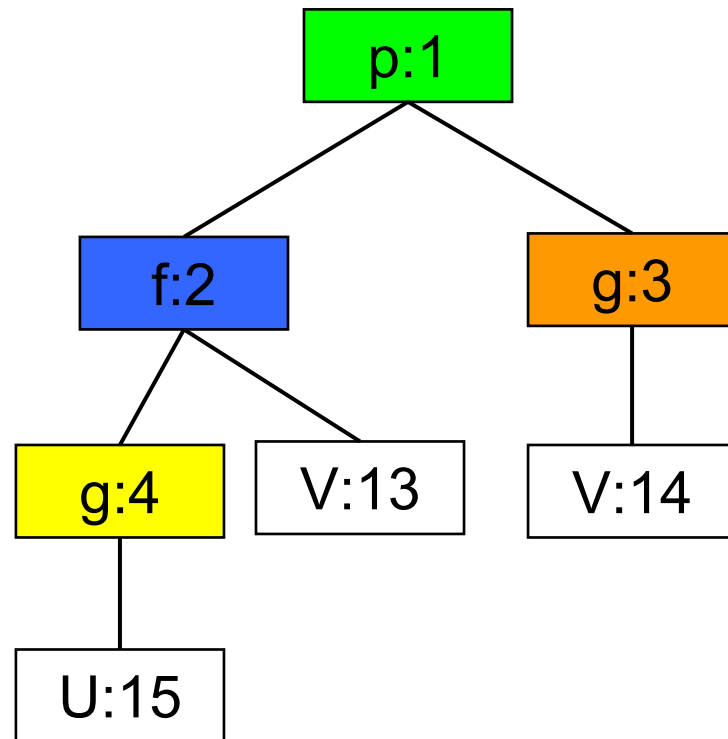
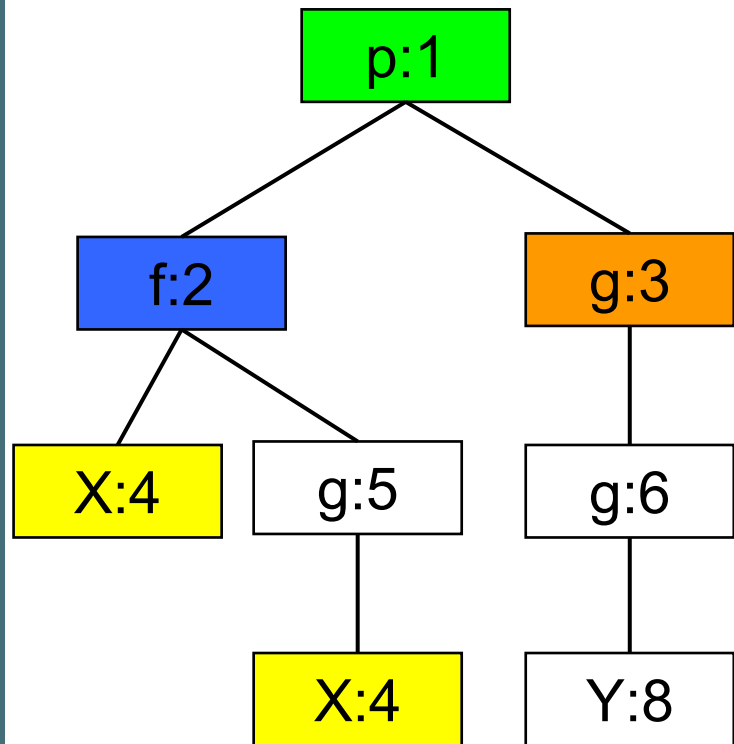
$$p(f(X, g(X)), g(g(Y))) = p(f(g(U), V), g(V))$$



Unification: Ullman (Huet?)

Example:

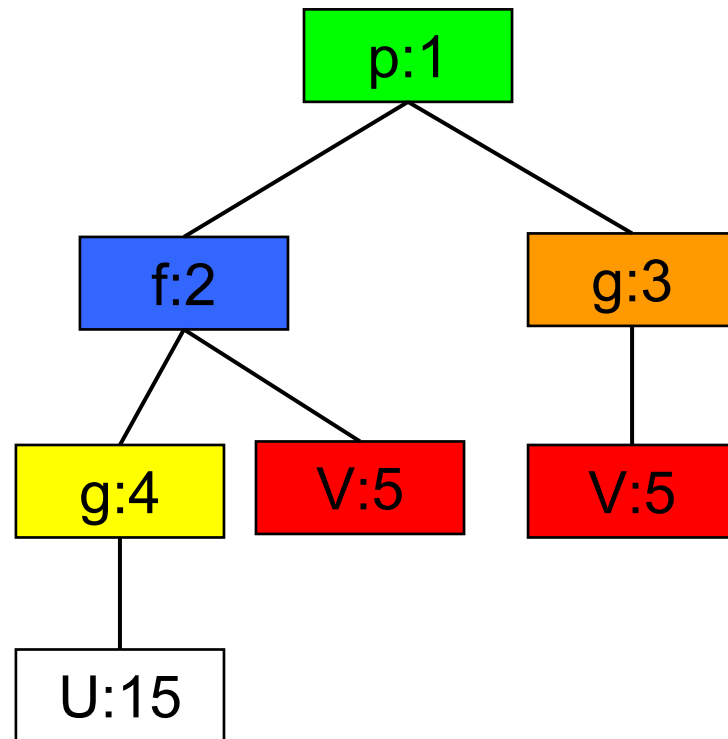
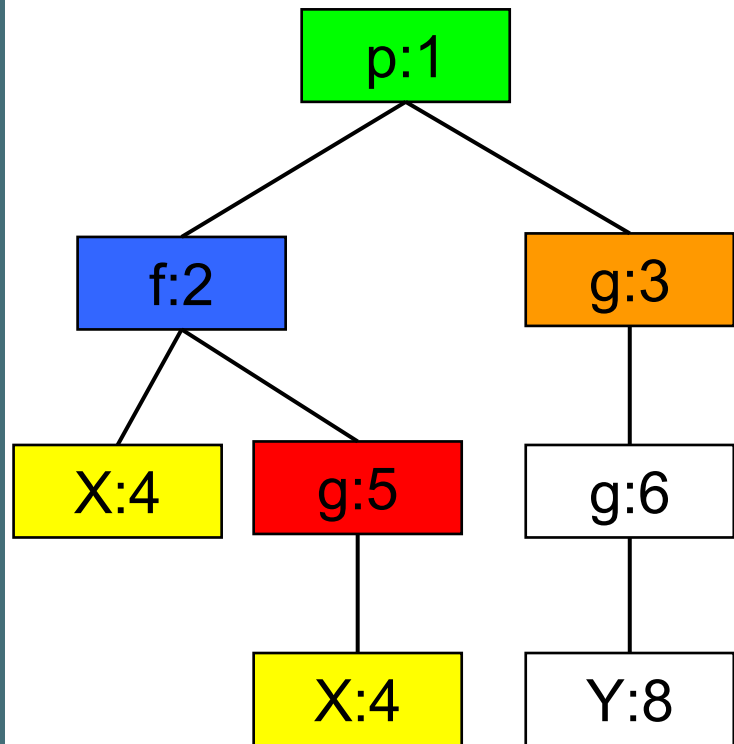
$$p(f(X, g(X)), g(g(Y))) = p(f(g(U), V), g(V))$$



Unification: Ullman (Huet?)

Example:

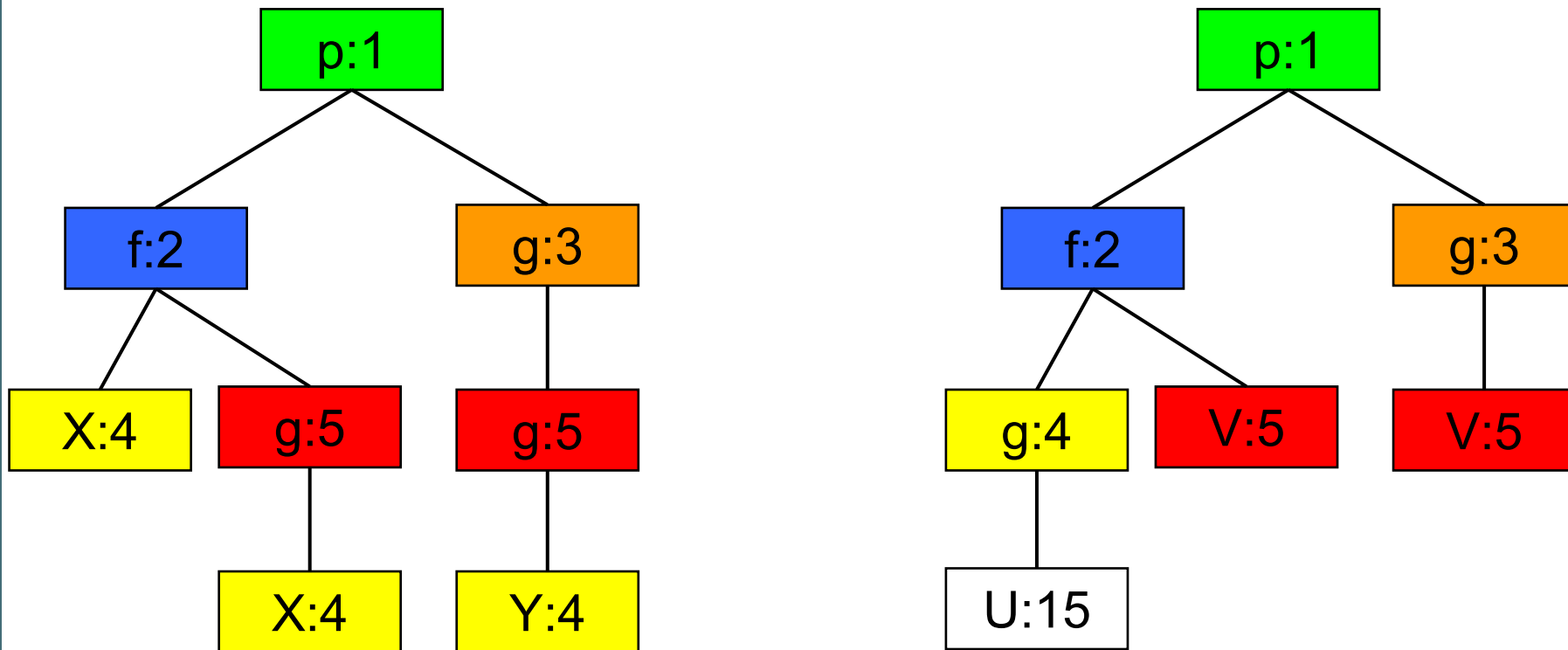
$$p(f(X, g(X)), g(g(Y))) = p(f(g(U), V), g(V))$$



Unification: Ullman (Huet?)

Example:

$$p(f(X, g(X)), g(g(Y))) = p(f(g(U), V), g(V))$$

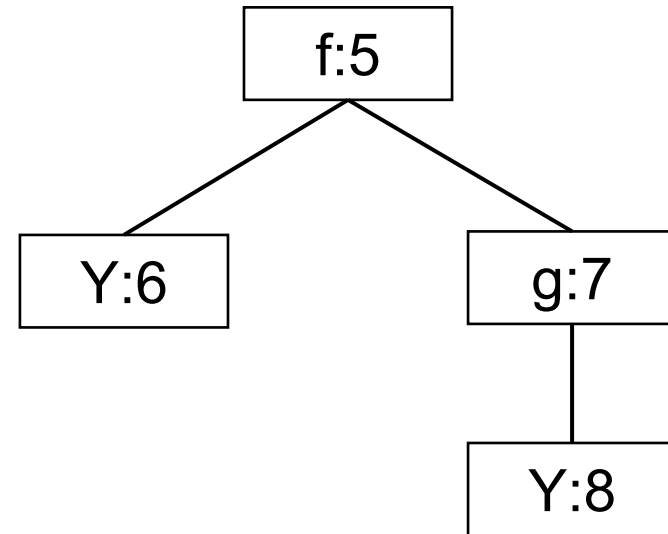
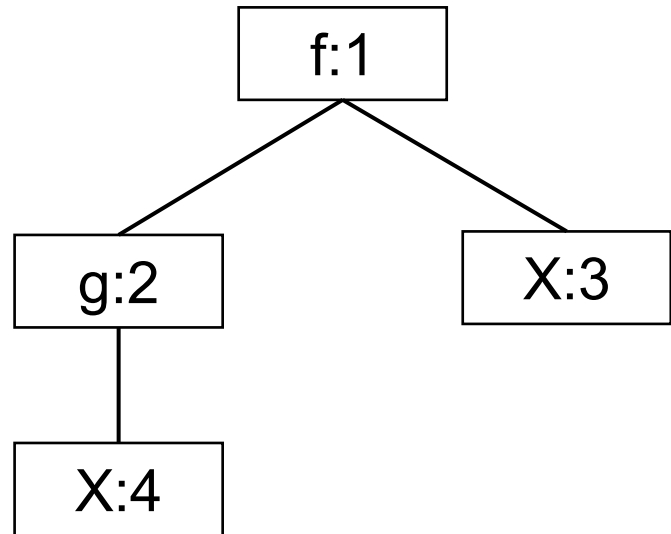


Construction of substitution (mgu): $Y \mapsto g(U)$, $X \mapsto g(U)$, $V \mapsto g(X)$

Explicit solution: $X = g(U)$, $Y = g(U)$, $V = g(g(U))$

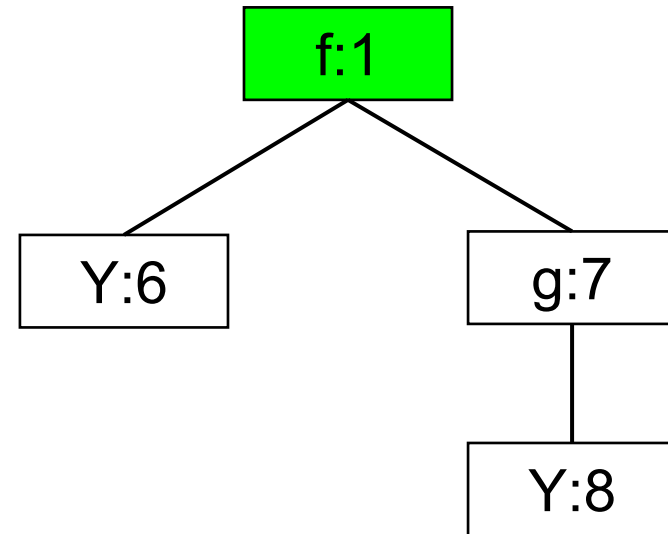
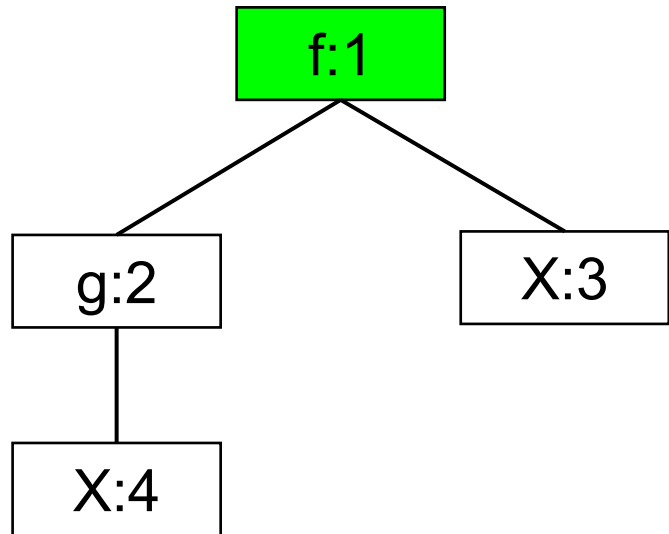
Unification: Ullman (Huet?)

$$f(g(X), X) = f(Y, g(Y))$$



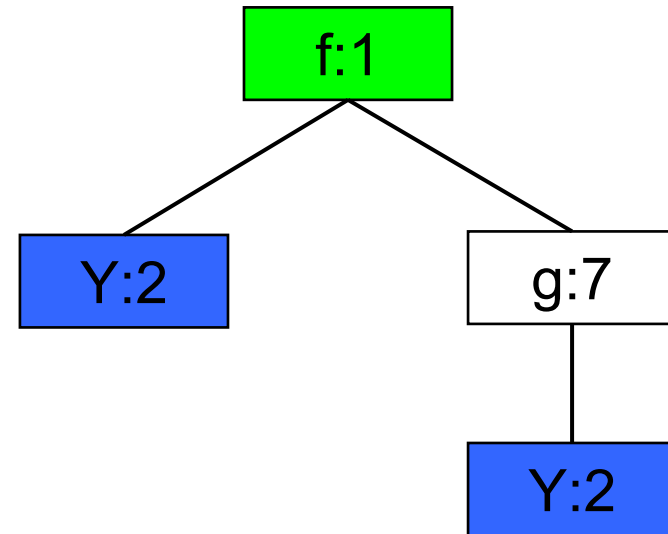
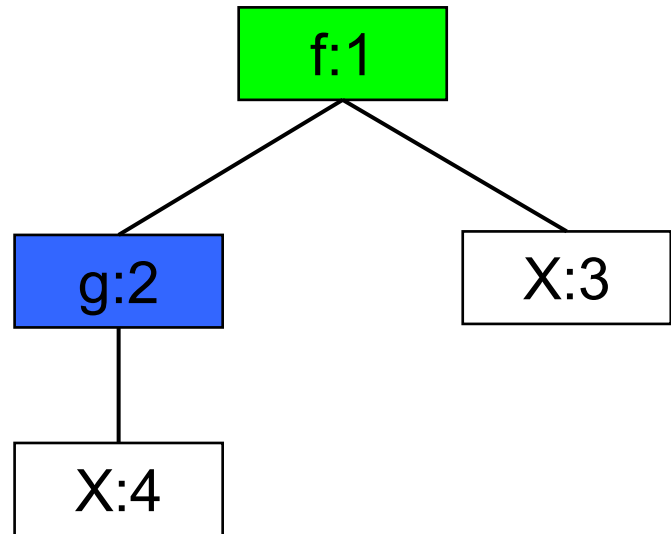
Unification: Ullman (Huet?)

$$f(g(X), X) = f(Y, g(Y))$$



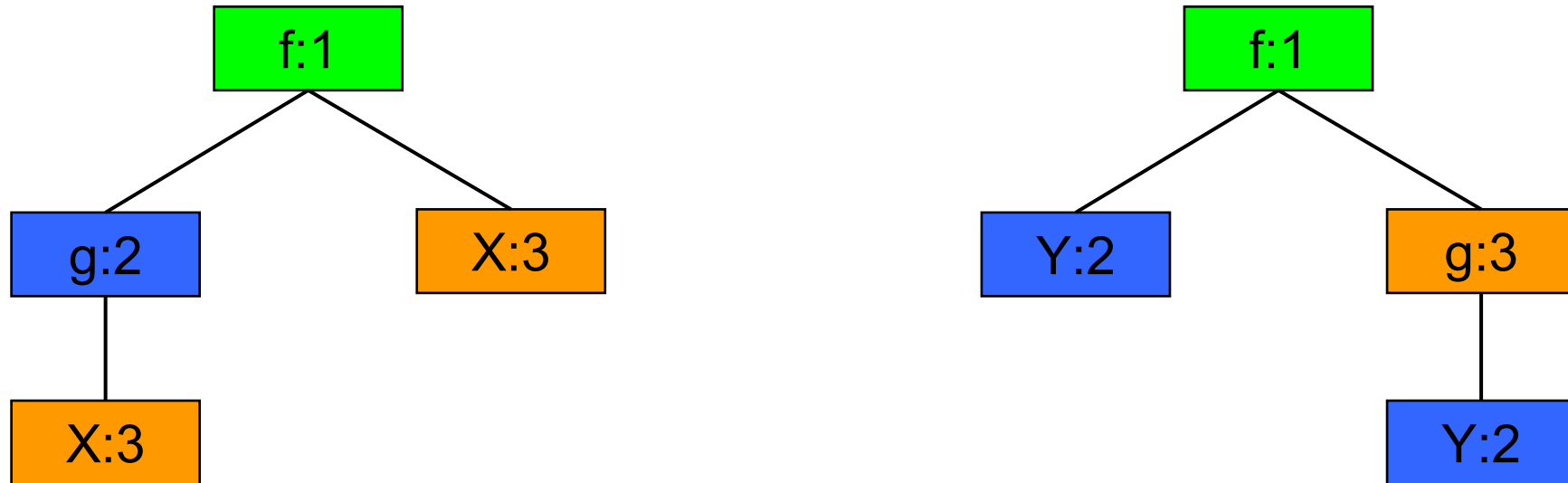
Unification: Ullman (Huet?)

$$f(g(X), X) = f(Y, g(Y))$$



Unification: Ullman (Huet?)

$$f(g(X), X) = f(Y, g(Y))$$



Construction of substitution (mgu): $Y \mapsto g(X)$, $X \mapsto g(Y)$ **CYCLE!**
Explicit solution does not exist: $X = g(g(\dots g(U)\dots))$, $Y = g(g(\dots g(U)\dots))$?

Unification: Analysis (Prívara, Ružička, 1989)

Asymptotical complexity of the algorithm:

- Traversing both terms creates equivalence classes such that each node is equivalent with itself. $O(n)$
- Synchronous traversal:
Find uzol n1
Find uzol n2
Union n1 n2
- Complexity of union-find task (amortised for n operations) is $O(n\alpha(n))$, where α is the inverse Ackermann function,
 $\alpha(D(n)) = n$
- Complexity of occurs-check test is $O(n)$ e.g. using topological sorting

$A(m, n) =$ if $(m == 0)$ return $n + 1$;
 else if $(n == 0)$ return $A(m - 1, 1)$;
 else return $A(m - 1, A(m, n - 1))$;

$D(n) = A(n, n)$

Unification: Implementation (Prívara, Ružička, 1989)

The unification algorithm can be implemented more efficiently than in previous slides (we intended to keep the analysis simple). **Tricks leading to an efficient implementation:**

- Initial phase can be skipped. Begin with the synchronous traversal. Maintain equivalence classes for nodes and equivalence classes for variables
- Whenever Find operation fails, establish new equivalence class
- In equivalence classes, besides nodes we remember also a functor and a variable. If a functor does not match the functor already stored in a class, then immediately terminate. If a variable does not match the variable already stored in a class, make these variables equivalent
- Occurs-check is modified: test whether a variable in the resulting substitution is equivalent to a variable on the left-hand side

Unification: explicit expression of the solution

The result of unification is a substitution (mgu)

Beware! An attempt to explicitly express the solution can lead to exponential complexity!

Example:

$$f(x_0, x_1, x_2, \dots, x_{n-1}) = f(g(x_1, x_1), g(x_2, x_2), \dots, g(x_n, x_n))$$

Solution:

$$x_0 \mapsto g(x_1, x_1), x_1 \mapsto g(x_2, x_2), x_2 \mapsto g(x_3, x_3), \dots, x_{n-1} \mapsto g(x_n, x_n)$$

Explicit solutions computed backwards:

$$x_{n-1} = g(x_n, x_n),$$

$$x_{n-2} = g(g(x_n, x_n), g(x_n, x_n)),$$

$$x_{n-3} = g(g(g(x_n, x_n), g(x_n, x_n)), g(g(x_n, x_n), g(x_n, x_n))) \dots$$

...

Lengths of terms: 1, 6, 16, 36, 76, 156, 316, ...

Length of i-th term: $L_i = 2L_{i-1} + 4$

Translation of Datalog to relational algebra

We want to use relational algebra to compute Datalog programs using the following mapping:

Datalog

predicate

predicate definition (\leftarrow)

constant (ground term)

,

not

recursion

Relational algebra

relation

assignment to a relation ($:=$)

constant string

join (\bowtie)

antijoin (\ltimes)

iterative fixpoint operator (Φ)

But how about terms which contain variables?

Terms in Datalog are used only in arguments of predicates (this includes built-in predicates such as “<”, “>”, “=”, “+”, “-”, ...)

Datalog to relational algebra: conversion of arguments

If an argument of a predicate is a single variable, it can be directly mapped to an attribute of a relation. However, arguments in a Datalog rule can be more complex, e.g.:

$p(f(X, g(X)) \leftarrow r(f(X, Y), r(g(f(Y, g(Y))))).$

Subgoal $r(f(X, Y))$ has **one argument**, which depends on **two variables**.

Subgoal $r(g(f(Y, g(Y))))$ has one **argument**, which **depends on one variable**.

Arguments of subgoals and the head have a structure determined by functors

How to compute join $r(f(X, Y), r(g(f(Y, g(Y))))$, if predicate r is mapped to a one-attribute relation R ? How to assign the result of the join to a one-attribute relation P corresponding to the predicate p ?

The solution is extension of the relational algebra with two conversion operators:

atov: „arguments to variables“

vtoa: „variables to arguments“

Datalog to relational algebra: conversion of arguments

atov: „arguments to variables“

Let G be m -ary relation corresponding to subgoal $g(a_1, \dots, a_m)$, with variables X_1, \dots, X_n . The contents of the variables must fit the structure of arguments of the subgoal. The result of $\text{atov}(g, G)$ is **n -ary** relation B . It is a selection (combined with a projection) of values from G which match arguments of the subgoal $g(a_1, \dots, a_m)$

```
relation atov(g, G) {  
    B =  $\emptyset$ ;  
    for (each tuple  $[t_1, \dots, t_m] \in G$ ) {  
        if (match( $g(a_1, \dots, a_m)$ ,  $g(t_1, \dots, t_m)$ )) {  
            /* let  $\rho$  denote the matching substitution */  
            B = B  $\cup$  [ $X_1 \rho, \dots, X_n \rho$ ];  
        }  
    }  
    return B;  
}
```

Datalog to relational algebra: conversion of arguments

vtoa: „variables to arguments“

Let $B(X_1, \dots, X_n)$ be **n-ary** relation resulting from the computation of the body of a rule $h(a_1, \dots, a_m) \leftarrow \langle \text{body} \rangle$. The contents of the relation B must fit the structure of the head $h(a_1, \dots, a_m)$. This can be done as

$H = \text{vtoa}(h(a_1, \dots, a_m), B)$. Values from B are simply substituted to the terms in the head

```
relation vtoa(p, B) {  
    H =  $\emptyset$ ;  
    for (each tuple  $t \in B$ ) {  
         $\sigma = \emptyset$ ;  
        for ( $i = 1; i < n; i++$ )  
             $\sigma = \sigma \cup [X_i \mapsto t_i]$ ;  
         $H = H \cup [t_1, \dots, t_m] \sigma$ ;  
    }  
    return H;  
}
```

Datalog to relational algebra: conversion of arguments

Back to the example

$p(f(X, g(X)) \leftarrow r(f(X, Y), r(g(f(Y, g(Y))))).$

Predicate r has corresponding one-attribute relation R , predicate p has a corresponding one-attribute relation P

Translation of the rule to relational algebra:

$P = \text{vtoa}(p(f(X, g(X)), \text{atov}(r(f(X, Y)), R) \bowtie \text{atov}(r(g(f(Y, g(Y)))), R)))$

For example, if $R = \{f(0, 1), g(f(1, g(1)))\}$, the computation proceeds as follows:

$B1(X, Y) = \text{atov}(r(f(X, Y)), R) = \{[0, 1]\}$ /* first record matches */

$B2(Y) = \text{atov}(r(g(f(Y, g(Y)))), R) = \{1\}$ /* second record matches */

$B(X, Y) = B1(X, Y) \bowtie B2(Y) = \{[0, 1]\}$ /* natural join */

$P(X) = \text{vtoa}(p(f(X, g(X)), B(X, Y)) = \{[f(0, g(0))]\}$ /* conversion to head */

Datalog to relational algebra: conversion of arguments

General scheme of computation of a safe Datalog rule (without negation)

Consider a rule $h \leftarrow g_1, \dots, g_k$.

Let G_1, \dots, G_k be relations corresponding to predicates of subgoals g_1, \dots, g_k

1. For each subgoal g_i compute relation $B_i = \text{atov}(g_i, G_i)$. (Relation B_i contains all values for variables of subgoal g_i , which satisfy the subgoal g_i .)
2. Compute $B = B_1 \bowtie \dots \bowtie B_k$. (Relation B contains all values for variables of the rule body, which satisfy all the subgoals.)
3. Convert the result to arguments of the head: $H = \text{vtoa}(h, B)$

We can yet optimise the computation in step 2 by early projecting out variables which appear neither in the rest of the join, nor in the head

Computation of programs without neg.: naïve iteration

- We can compute single rules. If there is more than one rule defining the same predicate, we compute the rules independently and then compute the union of the resulting relations
- If the predicate dependency graph is acyclic, topological sort determines the order in which predicates are computed
- If the predicate dependency graph is cyclic, we divide the predicates into extensional (given) and intensional. Relations corresponding to intensional predicates are initialised to an empty set and iterated according to Tarski fix-point theorem (each iteration computes new contents of the relations from their previous contents)

Naïve iteration can be perceived as follows: Consider a set of equations $\mathbf{P} = \mathbf{E}(\mathbf{P}, \mathbf{R})$, where \mathbf{P} is a set of intensional predicates and \mathbf{R} a set of extensional predicates. Initialisation: $\mathbf{P} = \emptyset$. Subsequently compute relations $\mathbf{P}^1, \mathbf{P}^2, \dots$ using the formula $\mathbf{P}^i = \mathbf{E}(\mathbf{P}^{i-1}, \mathbf{R})$. Finish when $\mathbf{P}^{i-1} = \mathbf{P}^i$. Tarski fix-point theorem guarantees that this process terminates (as natural join, union, projection and renaming in expressions \mathbf{E} are „monotone“ operations)

Computation of programs without neg.: naïve iteration

```
naive_iteration() {  
    for (all IDB predicates  $p_i$ )  
         $P_i = \emptyset$ ;  
    do {  
        change = FALSE;  
        for (all IDB predicates  $p_i$ ) {  
            old_ $P_i$  =  $P_i$ ;    /* old_ $P_i$  is  $P_i$  from previous iteration */  
            for (all rules  $r$  with head  $p_i(\dots)$ ) {  
                /* apply rule  $r$  */  
                 $P_i = P_i \cup$   
                vtoa( $p_i(\dots)$ ,  $\Pi_{\dots}(\text{atov}(\cdot, \cdot) \bowtie \text{atov}(\cdot, \cdot) \bowtie \dots)$ );  
            }  
            if (old_ $P_i$   $\neq$   $P_i$ )  
                change = TRUE; /* a relation has changed */  
        }  
    } while (change);  
}
```

Computation of programs without neg.: semi-naïve iter.

Differential scheme:

- Each intensional predicate p_i is assigned a difference Δp_i corresponding to a relation ΔP_i (initially an empty set)
- Individual rules are computed as in naïve iteration, but in the body of the rule of p_i , differences $\Delta p_i(\dots)$ are used instead of subgoals $p_i(\dots)$ (other subgoals are unchanged). Results are stored to both new ΔP_i as well as new P_i . However, only a difference against the old P_i is stored into ΔP_i :

$$\Delta P_i := \Delta P_i - P_i$$

$$P_i := P_i \cup \Delta P_i$$

- This is iterated until all ΔP_i computed in the last iteration are empty (i.e. no tuple was added)

This scheme can be yet significantly optimised by using an appropriate ordering of rules in the iteration (and by using additional differentials in the body of the rules). [R.Ramakrishnan, D.Srivastava, S.Sudarshan: Rule Ordering in Bottom-Up Fixpoint Evaluation in Logic Programs, 1990]

Computation of programs without neg.: semi-naïve iter.

```
seminaiive_iteration() {  
    for (all IDB predicates  $p_i$ ) {  
         $P_i = \emptyset$ ;  
         $\Delta P_i = \emptyset$ ;  
    }  
    do {  
        for (all IDB predicates  $p_i$ ) {  
            for (all rules  $r$  with head  $p_i(\dots)$ ) {  
                /* apply rule  $r$  */  
                 $\Delta P_i = \Delta P_i \cup$   
                 $\text{vtoa}(p_i(\dots), \Pi_{\dots}(\text{atov}(\cdot, \Delta \cdot) \bowtie \text{atov}(\cdot, \Delta \cdot) \bowtie \dots))$   
                 $- P_i$ ;  
                 $P_i = P_i \cup \Delta P_i$ ;  
            }  
            if ( $\Delta P_i \neq \emptyset$ )  
                change = TRUE; /* a relation has changed */  
        }  
    } while (change);  
}
```

Negácia v databázach a logickom programovaní

2. časť úvodnej prednášky

Negácia

- Ako zistiť, že niečo neplatí ako to dokázať?
- Ako zistiť, že teória je nekonzistentná?
- Bertrand Russell: Keď pripustíme jednu nepravdu dá sa dokázať všetko.
- Predikátový kalkul: Z množiny pozitívnych faktov a implikácii nevyplýva žiaden negatívny výrok.
- Databázisti a umelí inteligenti sú s matematickou logikou nespokojní.

O logike

- Logika je len jedna, kto myslí inak, myslí nelogicky.
– Aristoteles
- Matematická logika je od Boha a pre Boha.
– Tretej cesty niet: $A \vee \neg A = \text{true}$, Boh je vševědúci.
- **Neviem, možno, pravdepodobne** to sú výmysly diabla.
- Svetonázorové otázky
 1. Čo existuje ?
 2. Čo platí (je pravda) ?
 3. Vzťah k protirečeniam: $A \wedge \neg A = \text{false}$.
- Rôzne oblasti ľudskej vedy sa stavajú k svetonázorovým otázkam rôzne.

Krátky prehľad

- Empirické vedy (fyzika, medicína)
 - Každá hypotéza je pravdivá, pokiaľ ju niekto nevyvráti (neobjaví, fakty ktoré jej protirečia).
- Právo
 - Všetko platí, pokiaľ zákon neurčí inak.
 - Zákony si môžu protirečiť, pravidlá na riešenie protirečení
 - Lex superior derogat legi inferiori.
 - Lex posterior derogat legi priori.
 - Lex specialis derogat legi generali.
 - Ius summum saepe summa est iniuria. (Cicero)

Formálne systémy

- Propozicionálny kalkúl
 - Dvojhodnotový
 - Trojhodnotový a viachodnotové
- Predikátový kalkúl
 - prvého rádu (používame variantu s rovnosťou)
 - prvý a druhý rád nepodstatné rozšírenia I. rádu
 - vyšších rádov
- Lambda kalkúl
- Všetické „zvrhlé“ logiky (vyskytujú sa v UI).

Teórie a modely

- Teória = množina formúl (tvrdení).
- Modelom teórie nazývame akýkoľvek objekt, pre ktorý platia aspoň všetky tvrdenia teórie.
- Matematický model obsahuje doménu (obor, z ktorého premenné nadobúdajú hodnoty), relácie, ktoré sú priradené predikátovým symbolom a funkcie priradené funkčným symbolom
- Matematické modely teórie bez predikátových symbolov sú algebry
- Ak teória obsahuje aj predikátové symboly hovoríme o algebraických štruktúrach, alebo jednoducho o štruktúrach

Doménové nezávislé formuly

- Formula $\phi(\bar{X})$ je doménovo nezávislá, ak pre dve štruktúry $\mathfrak{I}1 = \langle D1, \mathcal{F}, \mathcal{R} \rangle$ a $\mathfrak{I}2 = \langle D2, \mathcal{F}, \mathcal{R} \rangle$, odlišujúce sa iba v doméne je množina $\{\bar{x}: \phi(\bar{x})\}$ rovnaká.
- Je nerozhodnuteľné, či formula ϕ je doménovo nezávislá.
- Ak formula ϕ je doménovo nezávislá, potom bezpečná formula $\Delta \wedge \phi$, je jej ekvivalentná.

Modely, algebry a databázy

- Vždy popísané formulami predikátového kalkulu
- Všeobecné modely (záujem matematikov)
 - veľké modely (ultra produkt)
 - spočítateľné modely (zostrojené z termov, Herbrandovské)
- Algebry sú o formuliach, ktoré sú splnené pre špecifické domény (grupa, pole, zväz,). Vlastne, axiomy algebry definujú tieto domény.
- Spočítateľná alebo konečná doména, s rovnosťou a prípadne usporiadaním. Doménovo nezávislé formuly vypovedajú o reláciach.

Ako je to v databázach

- V databázach:
 - teória = EDB + program
 - model = EDB + IDB
- Ak program neobsahuje negácie, počítame najmenší model naívnou (seminaívnou) iteráciou. Tarského veta zaručuje, že výpočet úspešne skončí.
- Sémantika najmenšieho modelu súhlasí s matematickou sémantikou.
 - V najmenšom modeli datalógového programu bez negácie platí práve to, čo vo všetkých modeloch tohto programu.

Tarskéhoho veta o pevnom bode

Úplný zväz: $S = \langle D, \sqsubseteq, \sqcup, \sqcap, \perp, \top \rangle$

\sqsubseteq čiastočné usporiadanie

\sqcup l.u.b., sup, join

\perp dolník

\sqcap g.l.b., inf, meet

\top horník

Každá neprázdna množina má l.u.b. (suprémum).

Veta: Nech F je zobrazenie z úplného zväzu S do S také, že $x \sqsubseteq y \Rightarrow F(x) \sqsubseteq F(y)$. Potom F má aspoň jeden pevný bod.

Dôkaz: Nech $U = \{x : x \sqsubseteq F(x)\}$. Množina U je neprázdna, $\perp \in U$.

Označme: $x_0 = \bigsqcup_{x \in U} x$. Pre každé $x \in U$ platí $x \sqsubseteq F(x) \sqsubseteq x_0$.

Preto aj $x_0 \sqsubseteq F(x_0)$ t.j. $x_0 \in U$. Preto aj $F(x_0) \sqsubseteq F(F(x_0))$ a $F(x_0) \in U$. Z toho ale plynie $F(x_0) \sqsubseteq x_0$. Teda $x_0 = F(x_0)$.

Výpočet rekurzívnych programov bez negácie

Systém rovníc: $\vec{P} = \vec{E}(\vec{P}, \vec{R})$, kde \vec{P} sú intencionálne a \vec{R} extenzionálne predikáty.

Riešenie: naivnou iteráciou. Začnememe $\vec{P}_0 = \vec{0}$. Postupne počítame postupne P_1, P_2, \dots , podľa vzorca:

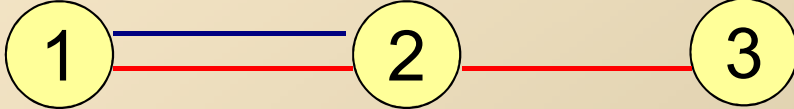
$$\vec{P}_i = \vec{E}(\vec{P}_{i-1}, \vec{R}),$$

pokiaľ $\vec{P}_i \neq \vec{P}_{i+1}$.

Podľa Tarského vety o pevnom bode tento proces vždy skončí (konverguje). Stačí overiť, že prirodzené spojenie, zjednotenie, projekcia a premenovanie sú „neklesajúce“ operácie.

Vypočítané riešenie je najmenší pevný bod uvedeného systému rovníc. Z vlastností implikácií plynie, že každé riešenie uvedeného systému rovníc musí obsahovať vety vypočítané našim algoritmom.

Datalógové programy s negáciou

- Program s negáciou nemusí mať najmenší pevný bod.
 $p \leftarrow q.$
 $p \leftarrow p, \neg q.$
- Eventuálne môže mať niekoľko minimálnych pevných bodov.
 $\text{modrácesta}(x, y) \leftarrow \text{modrá}(x, y)$
 $\text{modrácesta}(x, y) \leftarrow \text{modrá}(x, z), \text{modrácesta}(z, y)$
 $\text{červenýmonopol}(x, y) \leftarrow \text{červená}(x, y), \neg \text{modrácesta}(x, y).$
- EDB 
- Minimálne pevné body:
EDB + $\text{modrácesta}(1, 2) + \text{modrácesta}(2, 3)$
EDB + $\text{modrácesta}(1, 2) + \text{červenýmonopol}(2, 3)$
stratifikovaný pevný bod

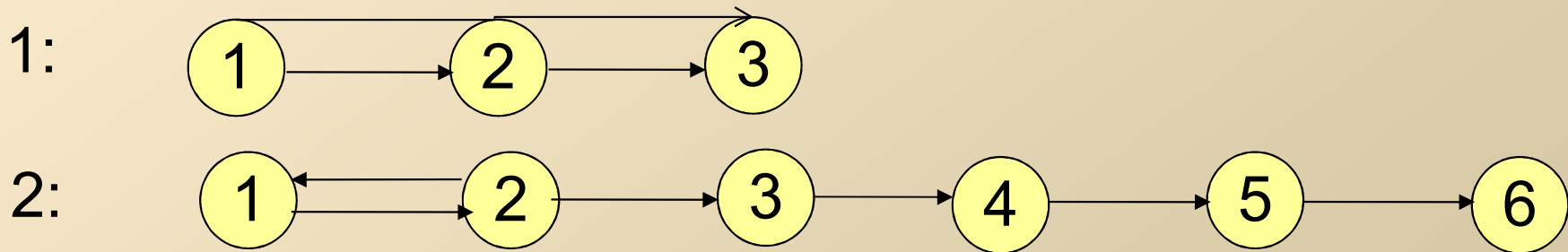
Zovšeobecnenia

Niekedy vieme vypočítať aj nestratifikované programy:

$$\mathbf{výhra(x)} \leftarrow \mathbf{t'ah(x, y), \neg výhra(y)}$$

To môže byť pravidlo pre nejakú hru (odoberanie zápaliek, nim), kde prehráva hráč, ktorý nemá ťah.

Príklady rôznych naplnení tabuľky ťah:



Oba prípady vieme vyriešiť. V každom prípade musíme pracovať s obsahom tabuliek. Odvoditeľnými uzavretými atómami (ground atoms).

Lokálne stratifikované modely

1. Vytvor graf závislostí IDB atómov:

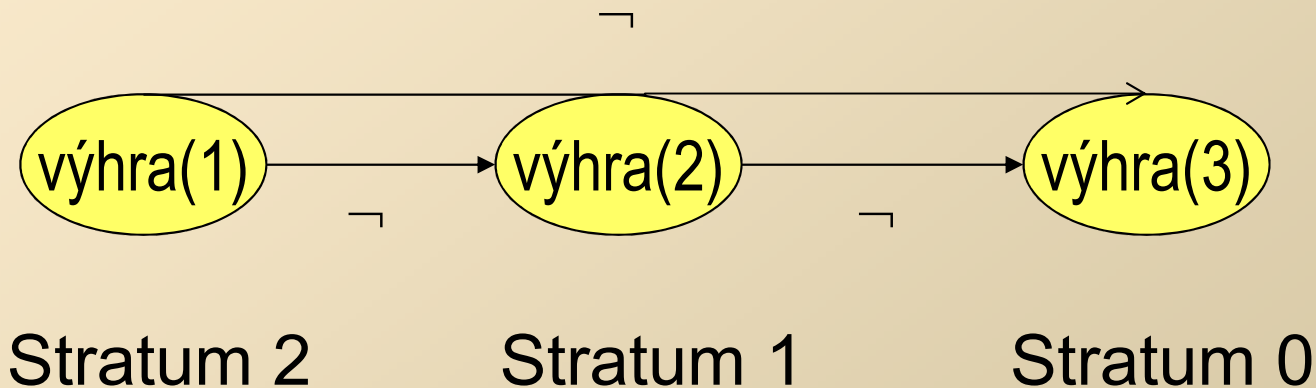
- Uzly sú relevantné IDB atómy.
- Hrana $p \rightarrow q$ práve vtedy, keď q sa vyskytuje instanciovanom tele pravidla s hlavou p .
- Značka (label) na hrane, ak q je negované.

2. Stratifikácia sa robí obvyklým spôsobom.

3. Model je lokálne stratifikovaný, ak počet úrovní (strata) je konečný.

Príklad

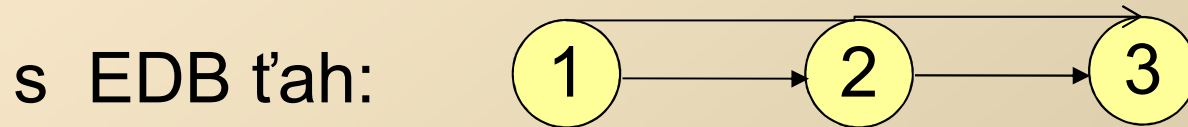
$\text{výhra}(1) \leftarrow \text{ťah}(1,2), \neg \text{výhra}(2)$
 $\text{výhra}(1) \leftarrow \text{ťah}(1,3), \neg \text{výhra}(3)$
 $\text{výhra}(2) \leftarrow \text{ťah}(2,3), \neg \text{výhra}(3)$



Stabílné modely (stable models)

- Hrubá intuícia: model je stabílny, keď aplikáciou programu sa nepodarí odvodiť nič nové, smie sa však využívať len negatívna informácia.
- Formálne: model M je stabílny, ak $M = \text{GLT}(M)$, kde GLT je Gelfond Lifschitzova transformácia.

Príklad: $\text{výhra}(x) \leftarrow \text{ťah}(x, y), \neg \text{výhra}(y)$



$M = \text{EDB} \cup \{\text{výhra}(1), \text{výhra}(2)\}$

je stabílny model. Žiadnou substitúciou za x, y neodvodíme $\text{výhra}(3)$.

Gelfond Lifschitzova transformácia (GLT)

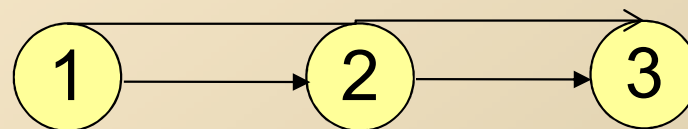
1. Inštancijuj pravidlá všetkými možnými spôsobmi.
2. Vynechaj inštanciované pravidlá s nepravdivým EDB alebo zabudovaným podcieľom (včítane negácie, \neg).
3. Vynechaj inštanciované pravidlá s IDB podcieľom $\neg p(\dots)$, kde $p(\dots)$ je v M .
4. Vynechaj podcieľ $\neg p(\dots)$, ak $p(\dots)$ nie je v M .
5. Vynechaj pravdivé EDB a zabudované podciele.
6. Použi zvyšné inštanciované pravidlá a EDB na odvodenie všetkých IDB uzavretých atómov.
7. Na záver pridaj EDB.

Príklad – znovu ten istý

$výhra(x) \leftarrow t'ah(x, y), \neg výhra(y)$

s EDB:

$M = EDB \cup \{výhra(1), výhra(2)\}$



1. krok (všetky inštanciované pravidlá)

2. krok (nemá efekt na dané pravidlá)

~~$výhra(1) \leftarrow t'ah(1, 2), \neg výhra(2)$~~ 3. krok $výhra(2) \in M$

~~$výhra(1) \leftarrow t'ah(1, 3), \neg výhra(3)$~~ 4. krok $výhra(3) \notin M$

~~$výhra(2) \leftarrow t'ah(2, 3), \neg výhra(3)$~~ 5. krok pravdivý EDB predikát

6. krok zostali nám dve pravidlá s prázdny (true) telom.

Teda odvodíme $\{výhra(1), výhra(2)\}$.

7. krok pridáme EDB. $GLT(M) = EDB \cup \{výhra(1), výhra(2)\}$

Trojhodnotové modely

- Trojhodnotový model pozostáva z:
 1. Z množiny pravdivých EDB faktov.
 2. Všetky ostatné EDB fakty sú považované za nepravdivé.
 3. Z množiny pravdivých IDB faktov.
 4. Z množiny nepravdivých IDB faktov.
 5. Pravdivostná hodnota zvyšných IDB faktov je neznáma (unknown).

Well-founded modely

- Začni s inštanciovanými pravidlami.
- Vyčisti „clean“ pravidlá t.j. vynechaj pravidlá s známými ne-pravdivými podcieľmi a vynechaj známe pravdivé podciele.
- Dva spôsoby odvodenia:
 1. „Obyčajný“: z pravdivého tela vyplýva hlava.
 2. „Unfounded sets“: predpokladá, že každý prvok unfounded set je nepravdivý.
- U je „unfounded set“ (pozitívnych uzavretých IDB atomov), ak každé zostávajúce inštanciované pravidlo s hlavou v U , má aj v tele prvok z U .
- Vlastnosť byť „unfounded set“ je uzavretá vzhľadom na zjednotenie. Teda existuje najväčšia unfounded set.
- Nikdy nedokážeme odvodiť pravdivosť prvku U , ale môžeme predpokladať jeho nepravdivosť.

Presnejšie „Unfounded set w.r.t. partial model“

Def: Nech M je čiastočný model pre program P .
Množinu U inštaciovaných (uzavretých) atómov nazývame unfounded set pre P vzhľadom k M .
Ak pre každé $A \in U$ a pre každé inštanciované pravidlo $A \leftarrow \text{body}$ z P platí:

- i.) body obsahuje podcieľ nepravdivý v M alebo
- ii.) nejaký pozitívny podcieľ z body je v U .

Lema: Ak U_1 a U_2 sú unfounded, potom aj $U_1 \cup U_2$ je unfounded.

Dôsledok: Existuje najväčšia unfounded set.

Konštrukcia well-founded modelu

Podciele z unfounded set sa nedajú dokázať (ani vyvrátiť).
Podľa princípu byrokracie chceme, aby ich negácia bola pravdivá.

repeat

- “clean” instantiated rules;

- make all ordinary inferences;

- “clean” instantiated rules;

- find the largest unfounded set and make its atoms false;

until no changes;

- make all remaining IDB atoms “unknown”;

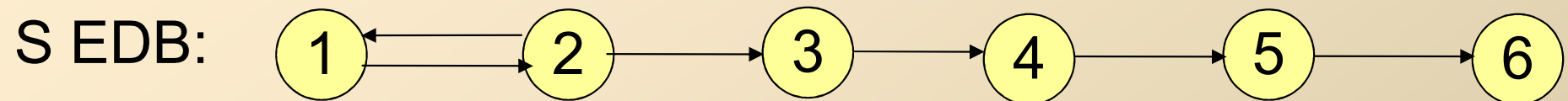
Alternujúci pevný bod

Alternating fixedpoint

1. Inštancijuj a „vyčisti“ pravidlá.
2. Na počiatku sú všetky IDB uzavreté atomy nepravdivé.
3. V každom ďalšom kole aplikuj GLT na EDB a na v predošlom odvodené pravdivé IDB uzavreté atomy.
4. Proces konverguje k alternácii dvoch množín pravdivých IDB faktov.
5. Nepárne kolá len zväčšujú; párne kolá len zmenšujú množinu pravdivých faktov.
6. V limite: pravdivé fakty sú stále pravdivé; nepravdivé fakty sú stále nepravdivé a fakty s pravdivostnou hodnotou „unknown“ alternujú.

Príklad – priama konštrukcia

$\text{výhra}(x) \leftarrow \text{ťah}(x, y), \neg \text{výhra}(y)$

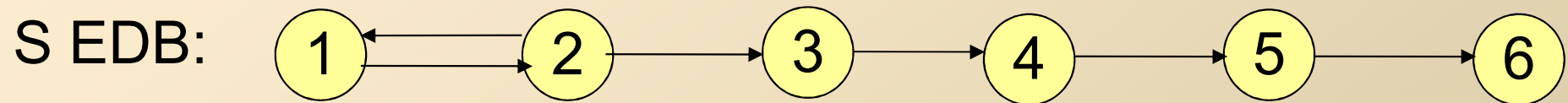


Inštanciované vyčistené pravidlá:

$\text{výhra}(x) \leftarrow \neg \text{výhra}(y)$	Kolo	G.U.S	pozitívne	negatívne
$\text{výhra}(1) \leftarrow \neg \text{výhra}(2)$	1.	{výhra(6)}		$\neg \text{výhra}(6)$
$\text{výhra}(2) \leftarrow \neg \text{výhra}(1)$	2.	{výhra(4)}	výhra(5)	$\neg \text{výhra}(4)$
$\text{výhra}(2) \leftarrow \neg \text{výhra}(3)$	3.	\emptyset	výhra(3)	
$\text{výhra}(3) \leftarrow \neg \text{výhra}(4)$				
$\text{výhra}(4) \leftarrow \neg \text{výhra}(5)$				
$\text{výhra}(5) \leftarrow \neg \text{výhra}(6)$				

Príklad – alternujúci pevný bod

$\text{výhra}(x) \leftarrow \text{ťah}(x, y), \neg \text{výhra}(y)$

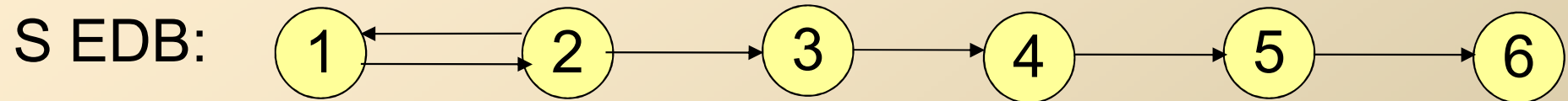


Inštanciované vyčistené pravidlá:

$\text{výhra}(1) \leftarrow \neg \text{výhra}(2)$	kolo	0	1	2	3	4	5
$\text{výhra}(2) \leftarrow \neg \text{výhra}(1)$	$\text{výhra}(1)$	F	T	F	T	F	T
$\text{výhra}(2) \leftarrow \neg \text{výhra}(3)$	$\text{výhra}(2)$	F	T	F	T	F	T
$\text{výhra}(3) \leftarrow \neg \text{výhra}(4)$	$\text{výhra}(3)$	F	T	F	T	T	T
$\text{výhra}(4) \leftarrow \neg \text{výhra}(5)$	$\text{výhra}(4)$	F	T	F	F	F	F
$\text{výhra}(5) \leftarrow \neg \text{výhra}(6)$	$\text{výhra}(5)$	F	T	T	T	T	T
	$\text{výhra}(6)$	F	F	F	F	F	F

Príklad – alternujúci pevný bod

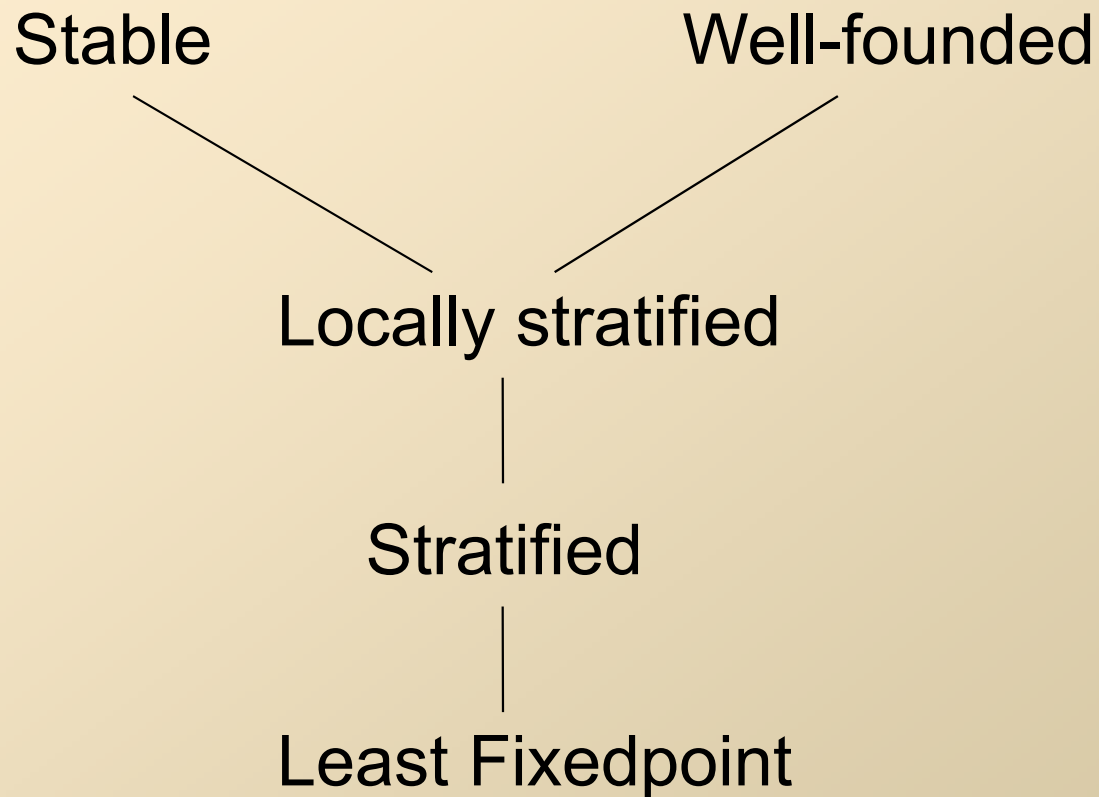
$\text{výhra}(x) \leftarrow \text{ťah}(x, y), \neg \text{výhra}(y)$



Inštanciované vyčistené pravidlá:

$\text{výhra}(1) \leftarrow \neg \text{výhra}(2)$	kolo	0	1	2	3	4	5
$\text{výhra}(2) \leftarrow \neg \text{výhra}(1)$	$\text{výhra}(1)$	F	T	F	T	F	T
$\text{výhra}(2) \leftarrow \neg \text{výhra}(3)$	$\text{výhra}(2)$	F	T	F	T	F	T
$\text{výhra}(3) \leftarrow \neg \text{výhra}(4)$	$\text{výhra}(3)$	F	T	F	T	T	T
$\text{výhra}(4) \leftarrow \neg \text{výhra}(5)$	$\text{výhra}(4)$	F	T	F	F	F	F
$\text{výhra}(5) \leftarrow \neg \text{výhra}(6)$	$\text{výhra}(5)$	F	T	T	T	T	T
	$\text{výhra}(6)$	F	F	F	F	F	F

Porovnanie sémantík



Inflačná sémantika

1. Na začiatku predpoladáme, že všetky uzavreté IDB atómy sú nepravdivé.
2. Opakujeme nasledujúci proces:
 - Inštaciu pravidiel použitím EDB a doteraz pravdivých IDB predikátov.
 - Ak je telo pravdivé (prítom predpokladáme, že všetky predikáty, ktoré nie sú pravdivé sú nepravdivé) je aj hlava pravdivá.
 - Pokiaľ sa niečo pridalo.
- Raz odvodený IDB fakt zostáva stále pravdivý,

Jednoduchý príklad

1. $p \leftarrow q.$

2. $q \leftarrow \neg r.$

3. $s \leftarrow \neg p.$

1. kolo: Všetky predikáty sú nepravdivé.

2. kolo: V dôsledku (2) q je pravdivé a v dôsledku (3) aj s je pravdivé.

3. kolo: Odvodíme p podľa (1).

4. kolo: Nič nové sa neodvodí. Koniec.

Inflačný model: $M = \{p, q, s\}.$

Negácia ako nekonzistencia

Negation as inconsistency

Daný je datalógový program P a jeho čiastočný model M (oboje chápeme ako množinu formúl). Formula $\neg\varphi$ platí, ak φ nie je konzistentné s $P \cup M$. (T.j. φ neplatí v žiadnom rozšírení $P \cup M$.)

Vyžaduje evidovať pozitívne aj negatívne atómy.

- Matematicky je to korektné v praxi slabé. (Chceme, aby platilo viac negatívnych faktov, ako sa dá dokázať.)
- D.M. Gabbay, H. J. Sergot: Journal of logic programming. Vol. 1, pp. 1-35 (1986).
- Podobá sa to na intuicionistickú logiku, alebo finite forcing (A. Robinson cca 1970).

DLV (TUV Vienna + University of Calabria)

- Dva druhy negácie

~ normálna databázova negácia

$\neg p$ intenzionálny predikát pre negáciu p

p^- EDB relácia pre negáciu p danú explicité.

Pravidlá: $\neg p \leftarrow p^-$.

$\neg p \leftarrow \sim p$. (princíp byrokracie)

Nemusíme posledné pravidlo použiť. Môžeme pridať aj iné pravidlá.

Požaduje sa aby platilo: $(\neg \exists \mathbf{x})(p(\mathbf{x}) \wedge \neg p(\mathbf{x}))$.

DLV (<http://www.dlvsystem.com/>)

- Disjunktívny datalóg v hlave pravidla môžu byť aj disjunkcie.
 - Napr. $p \vee \neg p \leftarrow$. Princíp vylúčenia tretieho.
- DLV počíta koherentné stabilné modely.
 - Model je koherentný, ak pre relácie zodpovedajúce všetkým predikátom platí:
 $p \cap p^- = \emptyset$.
- *Brave reasoning* - platí v aspoň jednom modeli.
- *Cautios reasoning* - platí vo všetkých modeloch.

O čom to je ? – Riešenie hry nim.

$\text{stav}(x) \leftarrow \text{ťah}(x,y)$
 $\leftarrow \text{ťah}(z,x)$
 $\text{prehra}(x) \leftarrow \text{stav}(x), \neg \text{ťah}(x, _)$ /* mat */
 $\leftarrow \text{stav}(x), \neg \text{dobrýťah}(x, _)$ /*nemá neprehrávajúci ťah*/
 $\text{dobrýťah}(x,y) \leftarrow \text{ťah}(x,y), \neg \text{výhra}(y)$
 $\text{výhra}(x) \leftarrow \text{ťah}(x, y), \text{prehra}(y)$
 $\text{remíza}(x) \leftarrow \text{stav}(x), \neg \text{výhra}(x), \neg \text{prehra}(x)$

Predikáty výhra, prehra a dobrýťah sú zacyklené.
Potrebujeme aspoň lokálne stratifikovaný pevný bod.
Aby sme sa zaobišli so stratifikovaným pevným bodom,
museli by sme využiť nejaké špecifické znalosti o hre.

Trade off: Zložitosť sémantiky (well-founded) za
pojmovú zložitosť programu.

Semijoin a antijoin

Definície:

Semijoin $R \bowtie S = \Pi_R(R \Join S)$

Antijoin $R \ltimes S = R - (R \bowtie S) = R - \Pi_R(R \Join S)$

Obe operácie sú odvodené. Dajú sa využiť na optimalizáciu a pri výpočte datalógových programov s negáciou.

Antijoin je zovšeobecnenie rozdielu aj na relácie s rôznymi schémami.

Zápis v datalógu:

$\text{semijoin}(X,Y,Z) \leftarrow r(X,Y,Z), s(Y, _).$

$\text{antijoin}(X,Y,Z) \leftarrow r(X,Y,Z), \neg s(Y, _).$

Vstupné množiny a pseudokľúče

Hoci domény môžu byť nekonečné spočítateľné množiny. Databázové relácie sú vždy konečné - tabuľky s konečným počtom riadkov.

Zaujíma nás, či dokážeme tabuľku vypísať.

Príklady nekonečných tabuliek (matematických predikátov):

- $=(x,y)$, $<(x,y)$, $y = \sin(x)$
- $\neq(x,y)$, $y = \arcsin(x)$
- $\text{perverse}(a,b,c,n) \Leftrightarrow a^n + b^n = c^n$ (a, b, c, n prirodzené čísla).

Definícia: Vstupnou množinou pre reláciu R , nazývame takú, množinu atribútov relácie R , že existuje len konečný počet riadkov R z danými hodnotami vo vstupnej množine.

Minimálna vstupná množina relácie R sa nazýva *pseudokľúč* relácie R .

Krajné prípady

- Pseudokľúč je prázdna množina (generátor)
 - databázové tabuľky
 - konečné matematické relácie
- Podmienka použiteľnosti je, že db-systém má k dispozícii program, ktorý postupne generuje (číta) prvky relácie.
- Pseudokľúč tvoria iba všetky atribúty relácie (rozpoznávač). Znovu db-systém musí disponovať programom, ktorý zistí, či daná n-tica patrí do relácie.

Dôsledok: Použiteľné sú len rekurzívne relácie.

Bezpečnosť programov s matematickými predikátmi

- Join a antijoin databazových relácii sú bezpečné operácie. Výsledok je bezpečná relácia.
- Join a antijoin bezpečnej relácie s nekonečnou reláciou je bezpečný pokiaľ sa „joinuje“ podľa vstupnej množiny nekonečnej relácie. Výsledok je bezpečná relácia.
- Projekcia, selekcia a agregácia aplikovaná na bezpečnú reláciu je bezpečná. Výsledok je bezpečná relácia.

Výpočet datalógových programov s negáciou (stratifikovaný pevný bod)

- Všetko robíme rovnako ako bez negácie jedine v kroku 2 pre výpočet pravidla v prípade nenegovaného predikátu generujeme join a v prípade negovaného predikátu antijoin.
- Anonýmne premenné (podčiarkovníky) postupne nahradíme čerstvými nikde sa nevyskytujúcimi premennými. Striktne vzaté nemali by sa vyskytovať v negovaných predikátoch (pravidlo nie je bezpečné), ale nevadia.
- Poradie negovaných a nenegovaných predikátov je dôležité antijoin sa nesmie použiť, pokiaľ v pozitívnej časti nie sú inštanciované spoločné premenné.

Výpočet zhora dole – SLD rezolúcia

Jedná sa o teoretický model (pure prolog).

Je to nedeterministický výpočet.

Dokazujeme konjunkciu cieľov G_1, \dots, G_n

1. Nedeterministicky vyber podcieľ (intenzionálny predikát napr. G_i).
2. Nedeterministicky pravidlo s hlavou vybraného podcieľu G_i
3. Unifikuj hlavu pravidla s vybraným podcieľom, pričom sa uprednostňujú premenné podcieľa.
4. Nahraď príslušný podcieľ telom pravidla.
5. Na novú konjunkciu aplikuj vypočítanú substitúciu (mgu).
6. Ak sú už všetky podciele extezionálne, over v databáze, či ich konjunkcia (kombinácia joinov a antijoinov) je neprázdna.

Skutočný prológ

- Relatívne nekorektná implementácia predošlej metódy.
- Nedeterminizmus je nahradený prehľadávaním do hĺbky (backtracking)
- Z unifikácie je vynechaná kontrola na výskyt
- Databáza je obmedzená na množinu extenziónálnych atómov zapamätateľnú v operačnej pamäti.
- Obsahuje rozšírenia skôr smerom k programovaniu než smerom k bázam dát.
- Vyššie uvedená metóda sa dá implementovať aj korektne, ale prológ si zo svojch chýb už urobil features.

Porovnanie metód zdola nahor (forward chaining) a zhora nadol (backward chaining)

Uvažujme program pre výpočet ciest v grafe.

$p(x, y) \leftarrow e(x, y).$

$p(x, y) \leftarrow e(x, z), p(z, y).$

Pokiaľ je našim cieľom vypočítať celý tranzitívny uzáver grafu, sa to zdá byť skoro jedno. Matching je jednoduchší ako unifikácia. Asi by som uprednostnil výpočet zdola nahor.

Ak však chceme zistiť, či existuje cesta $p(a, b)$, je výpočet zhora nadol evidentne výhodnejší a to aj v prípadoch, keď prehľadáme všetky cesty vychádzajúce z a (vedúce do b). Vďaka nekorektnej implementácii, prológ sa môže zacykliť a nenájst' riešenie dokonca z dvoch dôvodov:
Ľavá rekurzia (dá sa odstrániť), cyklus v dátach (v grafe).

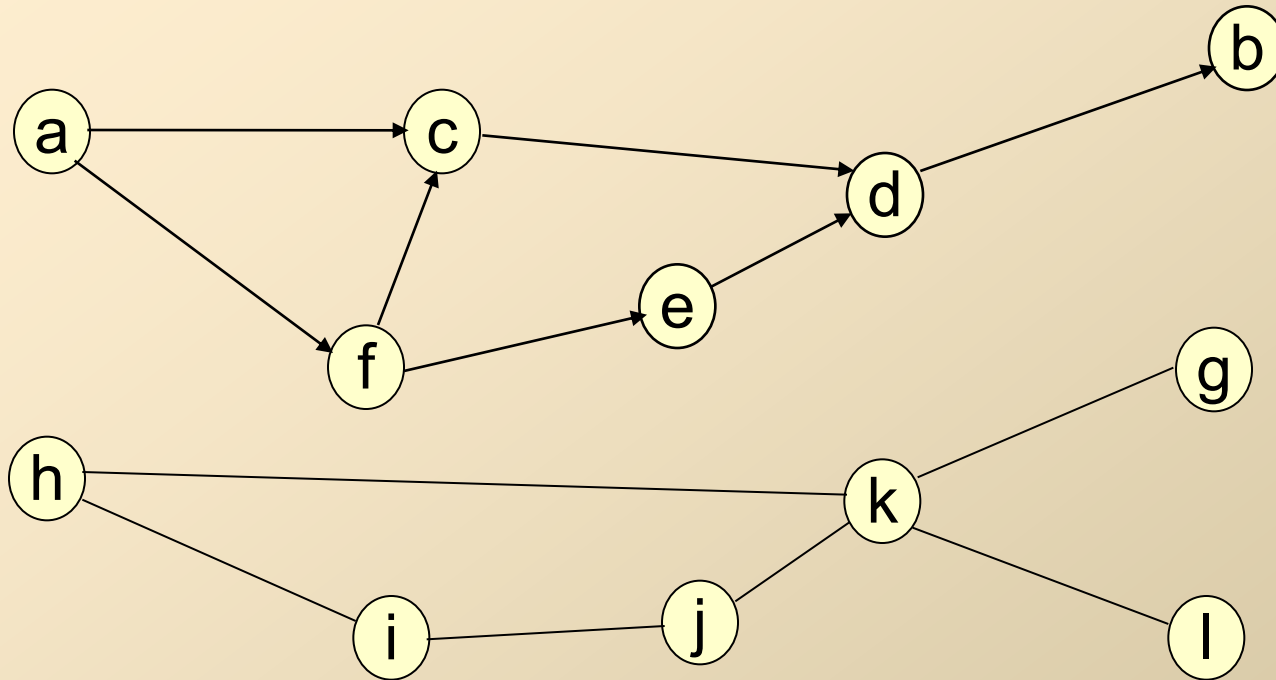
Výpočet logických programov zdola nahor

Ján Šturc

Datalóg a prológ

- Join $Q(x,y,z) = R(x,y) \bowtie S(y,z)$
 - $q(x,y,z) \leftarrow r(x,y), s(y,z).$
 - $q(X,Y,Z):- r(X,Y), s(Y,Z), \text{fail}.$
 - Datalóg počíta pomocou joinu. Prológ inštaciuje premenné pomocou SLD rezolúcie a tak postupne vypočítava n-tice joinu.
- Cesta v grafe
 - $p(x,y) \leftarrow e(x,y).$
 - $p(x,y) \leftarrow e(x,z), p(z,y).$
- Prológ vyžaduje program bez ľavej rekurzie. Datalóg s tým nemá problém. Na druhej strane uvažujme dotaz
?- p(a,b)

Príklad – inštancia grafu



Orientácia:

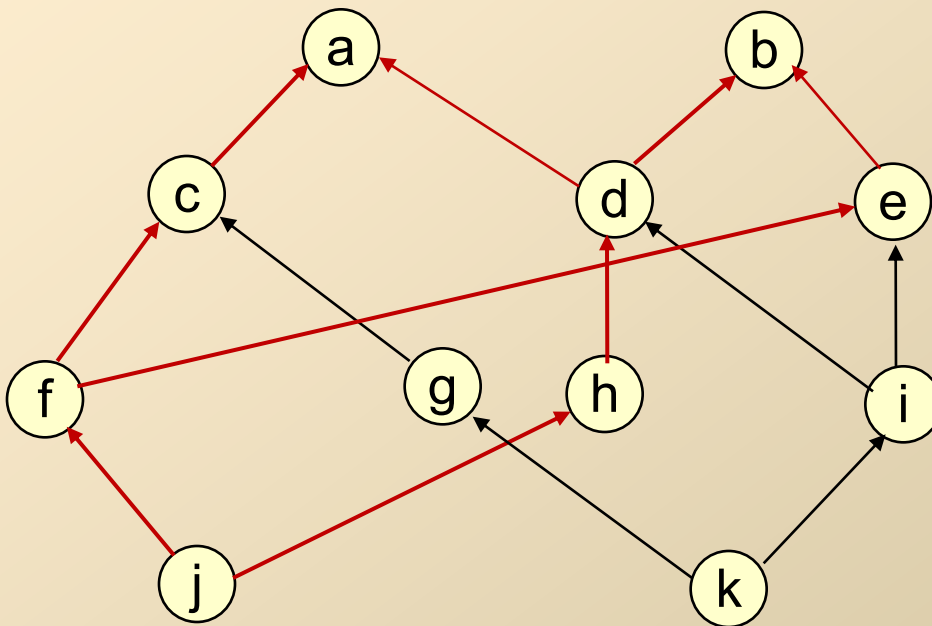
1. z ľava do prava

2. obojsmerná (neorientovaný graf)

Iný príklad – predkovia Johna

$r_1: \text{anc}(x,y) \leftarrow \text{par}(x,y)$ /* $\text{par}(x,y)$: y je rodič x, anc – predok */
 $r_2: \text{anc}(x,y) \leftarrow \text{par}(x,z), \text{anc}(z,y)$

$\text{Par} := \{ (c,a), (c,d), (d,b), (e,b), (f,c), (g,c), (h,d), (i,d), (f,e), (i,e), (j,f), (k,g), (j,h), (k,i) \}$

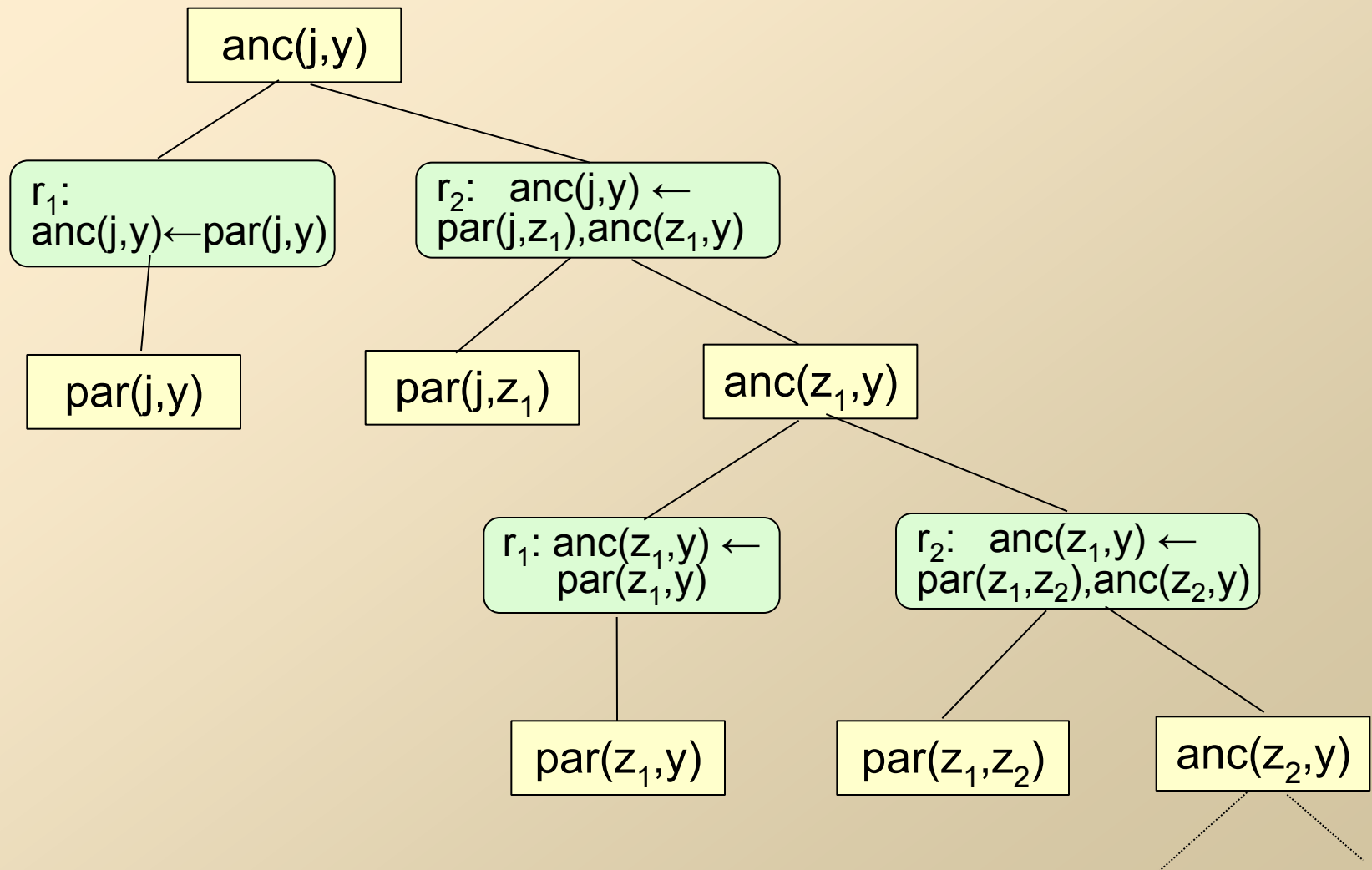


Môže existovať ľubovoľne komplikovaný graf pod vrcholami f, g, h, i, j, k neovplyvní to výpočet zhora dolu. Zťaží to ale metódu „buldozér“.

Strom pravidiel a cieľov (rule goal tree)

1. Koreň je cieľový uzol G_0 .
2. Synovia cieľového uzla sú pravidlové uzly, pre pravidlá ktorých hlava sa unifikuje s daným cieľom. Musíme ošetriť premenné (podobne SLD rezolúcii).
 - i. Pred unifikáciou premenné pravidla musia byť disjunktné s premennými cieľa.
 - ii. Pri unifikácii preferujeme premenné cieľa.
 - iii. Ak substituujeme za premennú v hlave, musíme tú istú substitúciu aplikovať aj na všetky výskyty premennej v tele pravidla.
 - iv. Premenné, ktoré sa nevyskytujú v hlave sú lokálne v tele pravidla. (Budme ich indexovať.)
3. Synovia pravidlového uzla sú podciele jeho tela. Na pomenovanie premenných sa vzťahuje bod 2.iii.

Príklad RGT pre predkov Johna



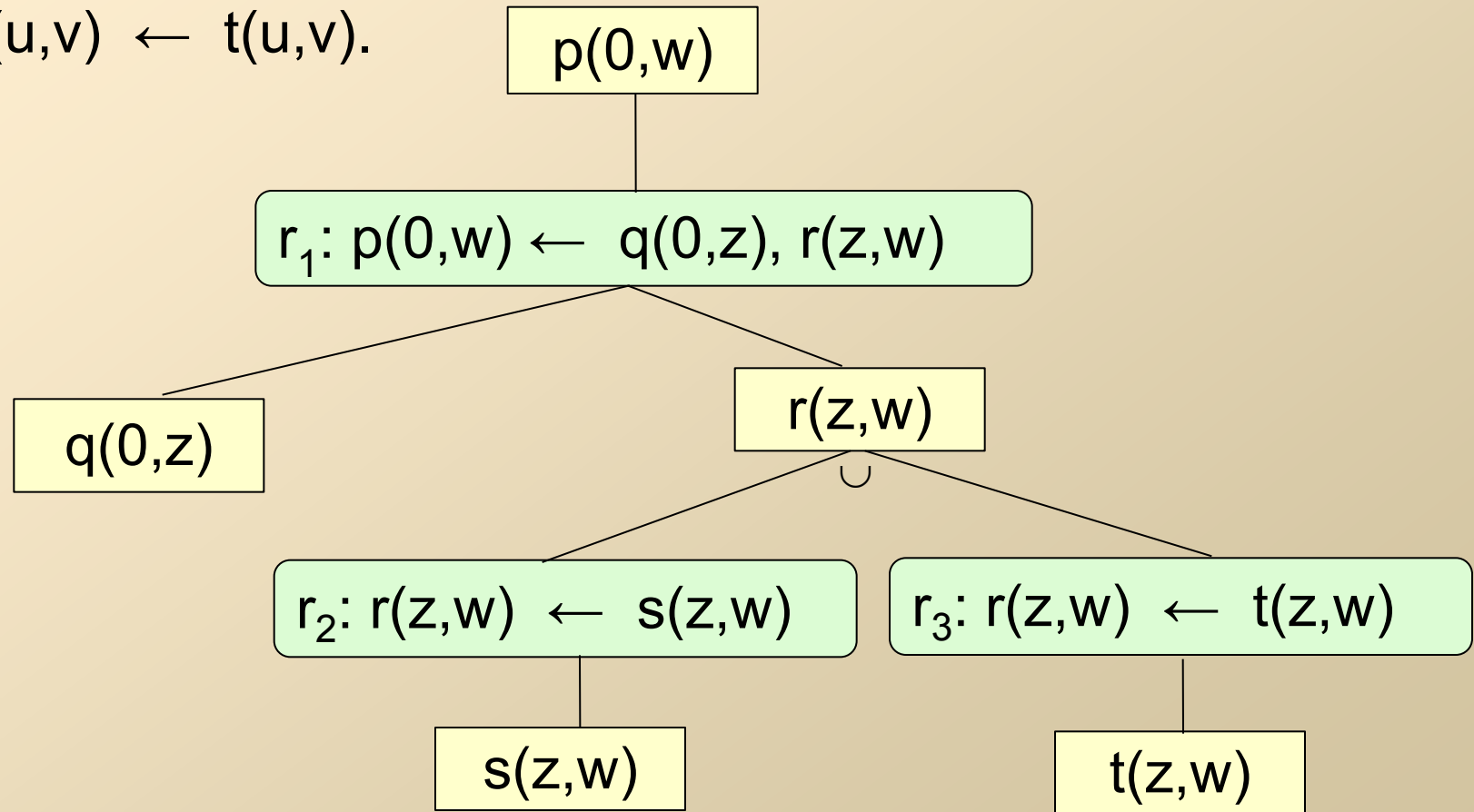
Nerekurzívny príklad

$r_1: p(x,y) \leftarrow q(x,z), r(z,y).$

$r_2: r(u,v) \leftarrow s(u,v).$

$r_3: r(u,v) \leftarrow t(u,v).$

Query: $?- p(0, w)$



Simulácia výpočtu zhora nadol – šírenie väzieb a výsledkov

1. S každým cieľovým uzlom je spojená väzobná relácia M („*magic predicate*“). Obsahuje argumenty, ktoré sú v tomto uzle viazané. Nadobúdajú konštatné hodnoty, alebo konečný počet hodnôt získaných predošlým výpočtom (odovzdané stranou, *sideway passed*).
2. V každom pravidlovom uzle sú pomocné (*supplementary*) predikáty S_0, S_1, \dots, S_i je spojené s i -tým podcieľom tela. Reprezentujú tie viazané premenné, ktoré sa presúvajú z hlavy, respektíve z predošlých podcieľov.
3. Oba druhy uzlov obsahujú výsledkovú reláciu. Pre cieľový uzol je to množina tých n -tíc, ktoré zodpovedajú cieľu a dajú sa odvodiť na základe podstromu uzla. Pre pravidlový uzol, sú to hodnoty pre hlavu.

Simulácia výpočtu zhora nadol – vlastný výpočet

1. Vlastný výpočet sa sústreďuje do pravidlových uzlov.
2. Vo vnútorných cieľových uzloch a koreni sa robí len zjednotenie výsledkov prichádzajúcich z ich synov.
3. Listy predstavujú „selekciu – join“ väzobného predikátu s EDB reláciou.
4. V pravidlových uzloch sa ciele synov najprv konvertuju na relácie nad premennými pomocou operácie ATOV. Vypočíta sa ich „join / antijoin“. Výsledok pre hlavu sa konvertuje pomocou operácie VTOA. Takto získaný výsledok sa pošle otcovi (nadradenému cieľovému uzlu).

Výpočet súčasne s vytváraním stromu pravidiel a cieľov

- Vstup: Množina bezpečných pravidiel, EDB a dotaz – cieľ G_0 .
- Výstup: Množina n-tíc (tuples), ktorá splňuje cieľ G_0 a vyplýva z danej množiny pravidiel a EDB.
- Metóda: Dve vzájomne rekurzívne procedúry.
 - *expandGoal*(M, G, R), kde G je cieľ, M väzobný predikát pre niektoré argumenty G . R je výsledná relácia.
 - *expandRule*(S_0, r, R), kde r je pravidlo, S_0 pomocný predikát, ktorý zaznamenáva niektoré viazané premenné pravidla r . R je opäť výsledná relácia.

procedure expandGoal(M, G, R)

```
{ if M =  $\emptyset$  then R :=  $\emptyset$ 
  else if G is a goal with an EDB predicate P then R := M  $\bowtie$  P
  else /* here G is a goal with an IDB predicate */
    { R :=  $\emptyset$ ; /* in R the result has to be accumulated */
    for each rule r whose head H unifies with G do
      {  $\tau$  := mgu(G, H);
        H' :=  $\Pi_M H\tau$ ; /*  $\tau(H)$  restricted to attributes of M */
        S0 := atov(H', M);
        expandRule(S0,  $\tau(r)$ , Rr);
        R := R  $\cup$  Rr
      }
    }
}
```


procedure expandRule(S_0 , r , R)

```
{ let  $r$  is  $H \leftarrow G_1, \dots, G_k$ ;  
  for  $i := 1$  to  $k$  do  
    {  $G_i' := \Pi_{S_{i-1}} G_i$ ; /*  $G_i$  restricted to arguments whose  
                                variables are attributes of  $S_{i-1}$  */  
      expandGoal( $M_i$ ,  $G_i$ ,  $R_i$ );  
       $Q_i := \text{atov}(G_i, R_i)$ ; /* convert  $R_i$  to variables */  
       $S_i := \Pi_T(S_{i-1} \bowtie Q_i)$ ; /*  $T$  is set of variables which  
                                   appear in  $S_{i-1}$  or  $Q_i$  and in the rule  $r$ . */  
    }  
   $R := \text{vtoa}(H, S_k)$   
}
```

Vlastnosti algoritmu

- Rekurzívny algoritmu expandGoal-expandRule je presnou simuláciou algoritmu zhora nadol a zacyklí sa práve vtedy, keď sa zacyklí backtracking.
- Oprava: QRGT (queue based rule goal tree expansion). Robiť algoritmus po úrovniach (do šírky). Dopĺňovanie väzobných, pomocných relácií a výsledku o nové n-tice sa takto dá urobiť.
- QRGT konverguje k pevnému bodu pre bezpečné datalógové programy.
- QRGT pre cieľ G a väzbu M_0 vygeneruje n -ticu $\mu = p(t_1, \dots, t_k)$ práve vtedy, ak
 - ju odvodí vyhodnocovanie zdola nahor pomocou tých istých pravidiel.
 - zodpovedá (match) cieľu G z viazanými atribútmi M_0 .

Ozdoby pre predikáty a pravidlá

- Ozdobou pre predikát je slovo v abecede $\{b, f\}$. Ozdoba hovorí, ktoré argumenty obsahujú len viazané premenné b a ktoré sú voľné f .
- Ozdoby využívame pri vyhodnocovaní pravidiel.
 - Premenná, ktorá sa vyskytuje vo viazanom argumente hlavy je ohraničená (nadobúda konečne mnoho hodnôt) pred vyhodnotením ľubovoľného podcieľa tela.
 - Premenná x je viazaná po vyhodnutí podcieľa G_i , ak premenná x bola viazaná pred vyhodnotením podcieľa G_i , alebo sa vyskytuje kdekoľvek v argumentoch G_i .
- Ozdoba pre pravidlo je: [\langle zoznam viazaných premenných \rangle | \langle zoznam voľných premenných \rangle].

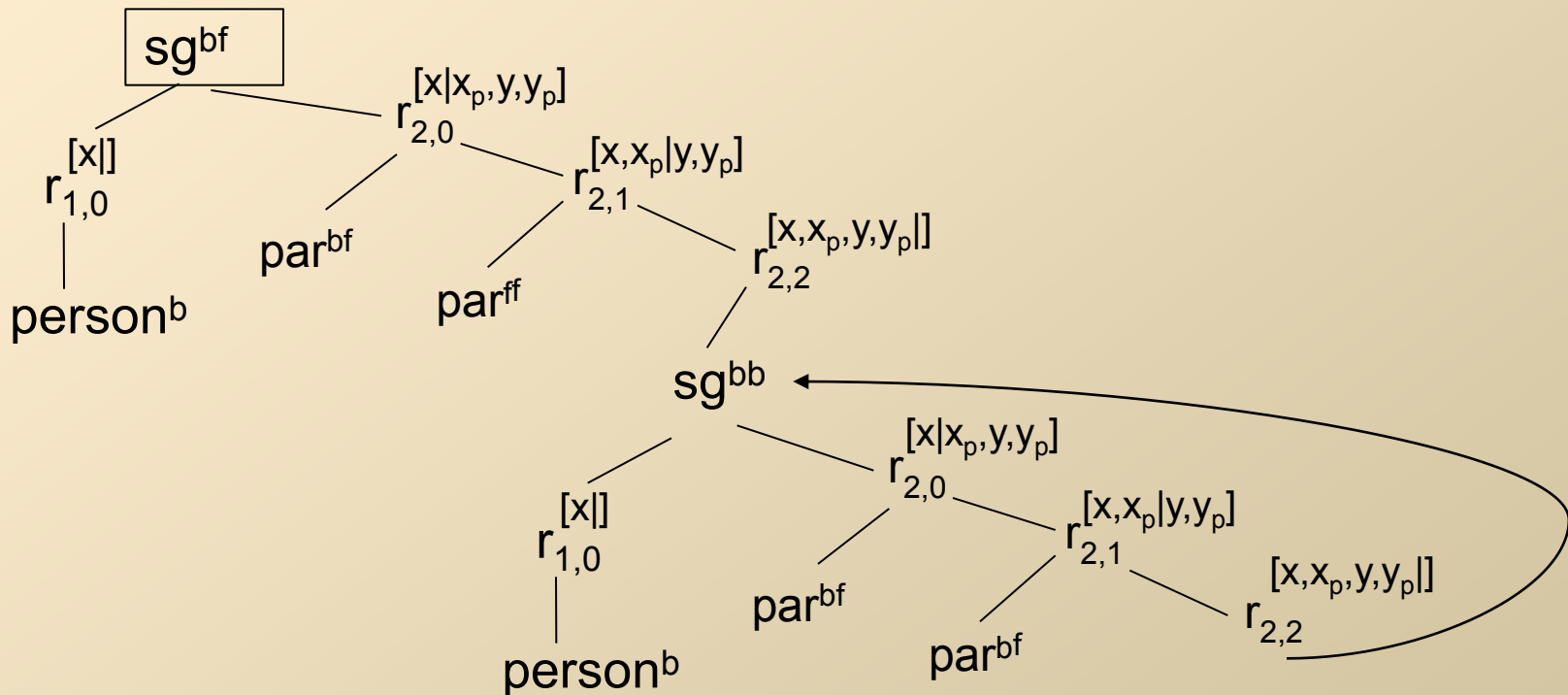
Graf cieľov a pravidiel – rule goal graph (RGG)

1. Cieľový uzol s EDB predikátom nemá výstupné hrany.
2. Z cieľového uzla s ozdobeným IDB predikátom p^α vedú hrany do pravidlových uzlov takých, že ich hlava je predikát p a majú viazané premenné, ktoré sa vyskytujú v argumentoch viazaných ozdobou α . $r_0^{[x_1, \dots, x_m | y_1, \dots, y_n]}$
3. Z pravidlového uzla $r_i^{[x_1, \dots, x_m | y_1, \dots, y_n]}$, $i \geq 0$, vedie hrana do
 - a) cieľového uzla q^β , kde q je i -tý predikát tela pravidla a β je ozdoba, ktorá viaže argumenty, ktoré obsahujú len premenné, ktoré sú viazané v tele pravidla.
 - b) ak i nie je posledný predikát, potom hrana do pravidlového uzla r_{i+1} , ktorý má navyše viazané všetky premenné vyskytujúce sa v argumentoch q .

Príklad – tá istá generácia (same generation)

$r_1: sg(x,x) \leftarrow person(x)$

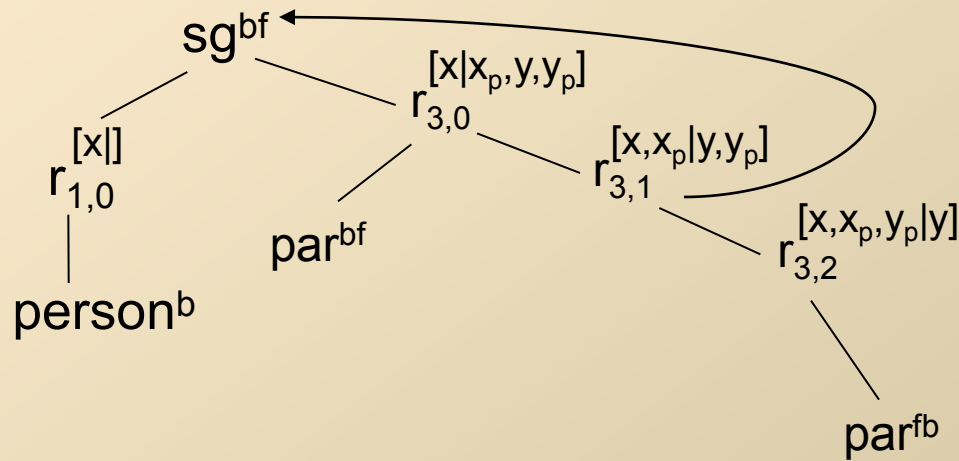
$r_2: sg(x,y) \leftarrow par(x,x_p), par(y,y_p), sg(x_p,y_p)$



Vylepšenie preusporiadaním podcieľov v pravidle

$r_1: \text{sg}(x,x) \leftarrow \text{person}(x)$

$r_3: \text{sg}(x,y) \leftarrow \text{par}(x,x_p), \text{sg}(x_p,y_p), \text{par}(y,y_p)$



Ako ďalej? Úpravy, ktoré pomáhajú

- Uniformita ozdôb = rovnaké väzby v pravidlách (making binding patterns unique)
- Rektifikácia podcieľov. Rektifikácia hláv pravidiel, ktorú robíme kvôli výpočtu pevného bodu nestačí. Problém je viacnásobný výskyt tej istej premennej v podcieti (*rectifying subgoals*).
- Preusporiadanie podcieľov v pravidlách (*reordering subgoals*).
- Nie každý graf cieľov a pravidiel sa dá výpočtovo realizovať (dovolené poradia ozdobených predikátov). Potrebujeme test realizovateľnosti (*the feasibility problem for rule-goal graph*).

Transformácia na program s uniformnými ozdobami

1. Pre každý ozdobený predikát p^α , vyskytujúci sa v RGG, vytvor nový predikát p_α .
2. Pre každé pravidlo s hlavou p , vytvor kópiu tohto pravidla r_α s hlavou p_α .
3. Vo všetkých uzloch daného pravidla r_0, r_1, \dots, r_k uprav synov (cieľové uzly)
 - a) Ak predikát q v uzle q^β je EDB predikát alebo zabudovaný predikát, ponechaj tento predikát v r_α tak, ako bol v r .
 - b) Ak predikát q v uzle q^β je IDB predikát, zmeň ho na q_β .

Príklad – tá samá generácia (slide 16)

r_{1_bf} : $sg_bf(x,x) \leftarrow person(x)$

r_{2_bf} : $sg_bf(x,y) \leftarrow par(x,x_p), par(y,y_p), sg_bb(x_p,y_p)$

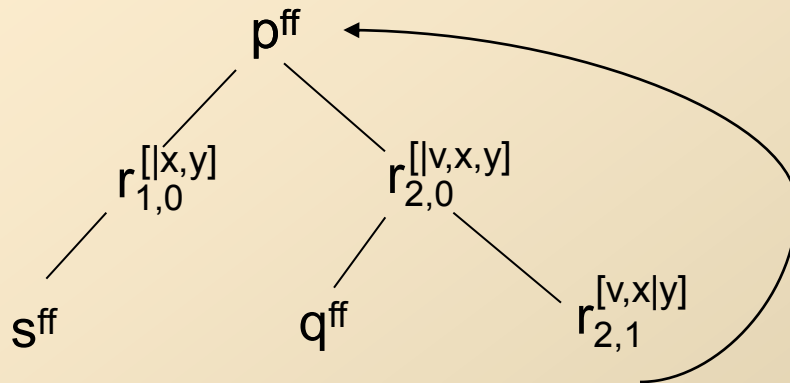
r_{1_bb} : $sg_bb(x,x) \leftarrow person(x)$

r_{2_bb} : $sg_bb(x,y) \leftarrow par(x,x_p), par(y,y_p), sg_bb(x_p,y_p)$

Iný príklad

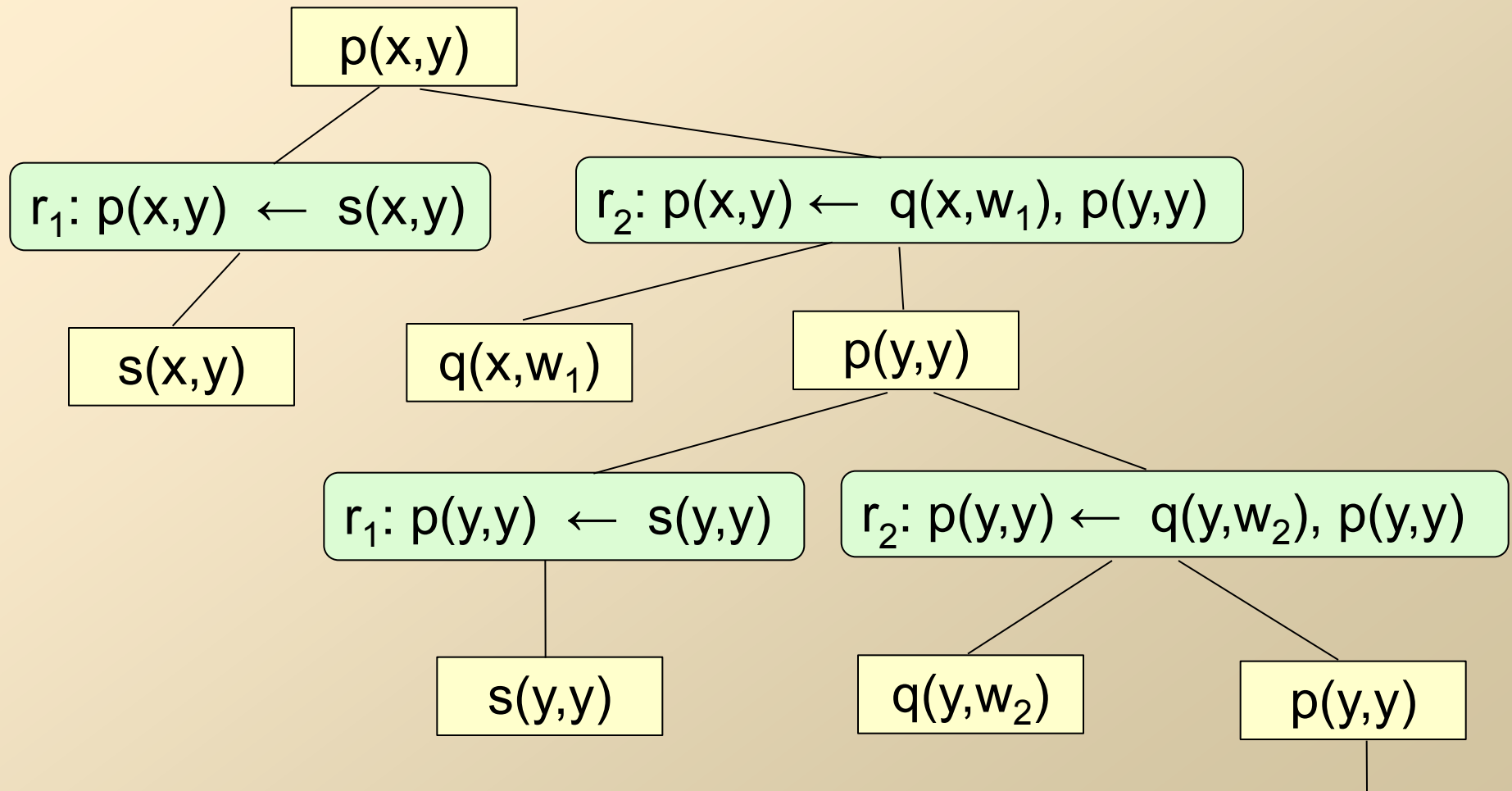
$r_1: p(x,y) \leftarrow s(x,y)$

$r_2: p(x,y) \leftarrow q(x,v), p(y,y)$



V tomto grafe je pravidle $r_{2,1}$ nesprávna ozdoba predikátu mala by byť p^{bb} . Spôsobilo to dvojnásobné použitie tej istej premennej y .

Strom cieľov a pravidiel pre predošlý príklad



Príklad pokračovanie pravidlá s rektifikovanými podcieľmi

$r_1: p(x,y) \leftarrow s(x,y).$

$r_2: p(x,y) \leftarrow q(x,v), p_1(y).$

$r_3: p_1(y) \leftarrow s(y,y).$

$r_4: p_1(y) \leftarrow q(y,v), p_1(y).$

Pre tieto pravidlá sa už dá urobiť „korektný“ graf cieľov a pravidiel. Bude pozostávať z troch podobných „poschodí“. Pre p^{ff} , p_1^f a p_1^b . Až „posledné poschodie“ sa zacyklí.

Algoritmus rektifikácie podcieľov

while existuje podcieľ g s opakovanými argumentami alebo konštantami **do** {

1. Nech g je podcieľ s predikátom p a opakovanými argumentami alebo konštantami vytvor nový predikát p' , odstránením opakovaných argumentov a konštant.
2. Pre každé pravidlo r s hlavou p vytvor nové pravidlo tak, že vypočítaš $\tau = \text{mgu}(g, p)$, pričom sa uprednostňujú premenné podcieľa. Nové pravidlo dostaneme z $r\tau$ tak, že všetky predikátu p s argumentami g nahradíme predikátom p' .
3. Nahrad' v pôvodných pravidlách podcieľ g podcieľom p' }

Tento proces musí skončiť. V každej iterácii sa zmenší počet argumentov v nejakom podcieli. Navyše ak pôvodné pravidlá boli bezpečné aj výsledok je bezpečný.

Algoritmus: Zuniformnenie ozdôb – making binding patterns unique

Vstup: Množina pravidiel a cieľ (dotaz).

Výstup: Modifikovaná množina pravidiel zabezpečujúca, že v strome cieľov a pravidiel má každý výskyt IDB predikátu rovnaké väzby. (V RGG tie isté ozdoby).

Metóda:

1. Rektifikuj podciele v pravidlách
2. Transformuj na program s uniformnými ozdobami (slide 19)

Usporiadanie podcieľov

1. Zabudované predikáty sa obvykle nedajú „vyhodnotiť“ pokiaľ argumenty „vstupnej množiny“ nie sú viazané.
2. Niekedy môžeme aj pre EDB predikáty preferovať takýto prístup napr. z dôvodu indexu, alebo keď EDB predikát nie je definovaný tabuľkou ale programom.
3. V prípade IDB predikátov definovaných pomocou funkčných symbolov napr. $\text{nat}(x)$. Nedokážeme nat^f .
4. Negované predikáty. Všetky argumenty musia byť viazané. (Túto podmienku môžeme trochu oslabiť, len na premenné, ktoré sa vyskytujú aj inde v pravidle. Aj tak pravidlo $p(x) \leftarrow q(x), \neg r(y)$ je problém. [Prelož do SQL.](#))

Dôsledkom je, že predikáty v tele pravidiel a RGG nemôžu byť v ľubovoľnom poradí. Niektoré poradia sú „lepšie“ ako iné.

Predpoklad viazané je lepšie – the bound is easier assumption

Definujeme čiastočné usporiadanie ozdôb: $\alpha \leq \beta$, ak β obsahuje b aspoň na tých miestách, kde ich má α .

Predpokladáme: ak $\alpha \leq \beta$ a vieme vyhodnotiť p^α , potom vieme vyhodnotiť aj p^β a vyhotenie p^β nebude výpočtovo náročnejšie ako p^α .

Ozdobu α pre, ktorú vieme predikát vyhodnotiť nazývame dovolenou.

Dôsledky uvedených definícií:

1. Pre každý predikát existuje množina minimálnych dovolených ozdôb.
2. Poradie predikátov môžeme vyberať pažravou (greedy) metódou.

Usporiadanie podcieľov v pravidle

Nech je dané:

- Pravidlo $H \leftarrow G_1, \dots, G_m$
- Ozdoba pre hlavu H
- Dovolené ozdoby pre predikáty tela

Chceme nájsť poradie ozdobených predikátov v tele, že je:

- **dovolené**, t.j. každý predikát má dovolenú ozdobu, a
- **korektné**, t.j. aspoň argumenty predpísané ozdobou sú v danom mieste výpočtu viazané.

Algoritmus **A**:

Udržujeme si množinu viazných premenných a v každom bode výpočtu testujeme, či existuje predikát, ktorý má dovolenú ozdobu z danej množiny. Ak neexistuje, pravidlo s danou hlavou sa nedá realizovať.

Bez ďalších ohraničení je to backtracking, ale za predpokladu „bound is easier“ sa nikdy nemusíme vracat’.

Algoritmus na testovanie realizovateľnosti RGG

Vstup: Množina pravidiel. Cieľ (dotaz) p^α , zoznam dovolených ozdôb pre EDB predikáty.

Výstup: RGG pre p^α , ak existuje, inak správa, že nexistuje.

Metóda: Budeme používať algoritmus A a predpoklad „bound is easier“. Zavedieme dve množiny: Množinu F zakázaných ozdobených cieľov. Na počiatku obsahuje všetky EDB predikáty s ozdobami, ktoré nie sú dovolené. Množinu I zaujímavých IDB cieľov. Na počiatku obsahuje len p^α . Následne použijeme algoritmus A na nejaký cieľ v množine I. Ak sa nám nepodarí nájsť realizovateľné usporiadanie tela niektorého pravidla presunieme tento cieľ do množiny F. Ak algoritmus A pre nejaké pravidlo uspeje, vložíme do množiny I požadované IDB ciele.

Základný cyklus algoritmu

```
for each adorned goal  $q^\beta \in I$  do  
  for each rule  $r$  with the head  $q$  do  
  {  $A(r, q^\beta, F)$ ; /* poradie podcieľov v  $r$  */  
    if an ordering found then  
      add to  $I$  adorned IDB predicates in selected order  
    else move  $q^\beta$  from  $I$  to  $F$ ;  
  }
```

Za predpokladu „bound is easier“ môžeme F reprezentovať maximálnymi prvkami t.j. predpokladať, že s každým prvkom obsahuje všetky menšie. Všetky väčšie prvky sú dovolené.

Algoritmus skončí keď

1. presunieme p^α do F (neúspech, neexistuje feasible RGG))
2. základný cyklus nezmení I (všetky pravidlá sa dajú realizovať s ozdobami v I .)

Príklad – tá istá generácia

$r_1: sg(x,x) \leftarrow person(x)$

$r_2: sg(x,y) \leftarrow par(x,x_p), par(y,y_p), sg(y_p,x_p)$

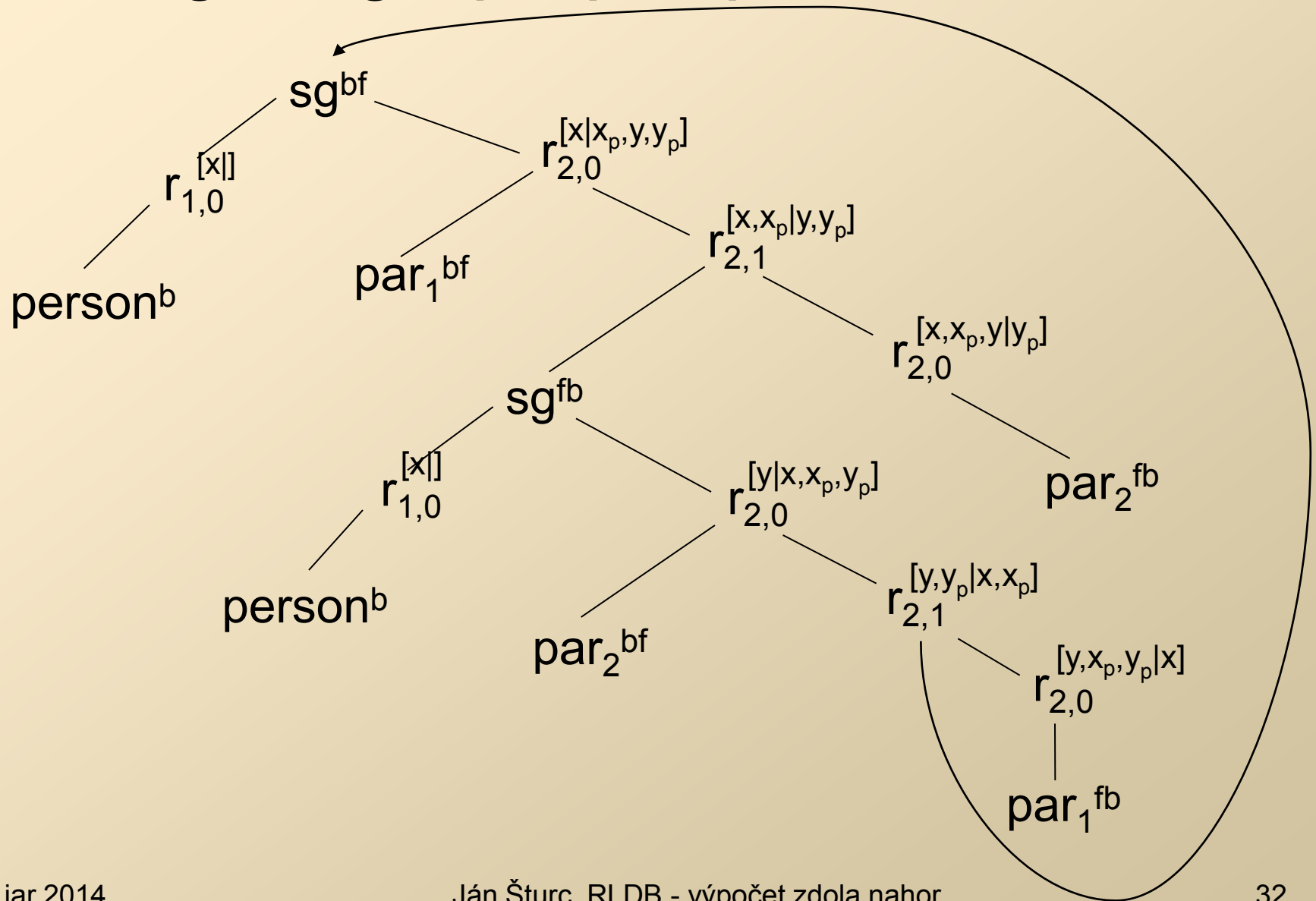
cieľ: $sg(john, w)$

$I = \{sg^{bf}\}; F = \{par^{ff}, person^f\}$

Pre r_1 je jediné možné usporiadanie s ozdobou b $person^b$.

Pre r_2 vyberie poradie tela $par_1^{bf}, sg^{fb}, par_2^{fb}$. V dôsledku pridá sg^{fb} do I . V nasledujúcom kole r_1 dopadne rovnako a r_2 pre sg^{fb} bude $par_2^{bf}, sg^{bf}, par_1^{fb}$. Žiadne nové ozdobené ciele sa nepridajú. $I = \{sg^{bf}, sg^{fb}\}$.

Rule goal graph pre príklad



Najkratšia cesta (negácia)

$r_1: p(x,y,d) \leftarrow e(x,y,d).$

$r_2: p(x,y,d) \leftarrow sp(x,z,d_1), e(z,y,d_2), d=d_1+d_2.$

$r_3: sp(x,y,d) \leftarrow p(x,y,d), \neg p(x,y,c), c < d.$

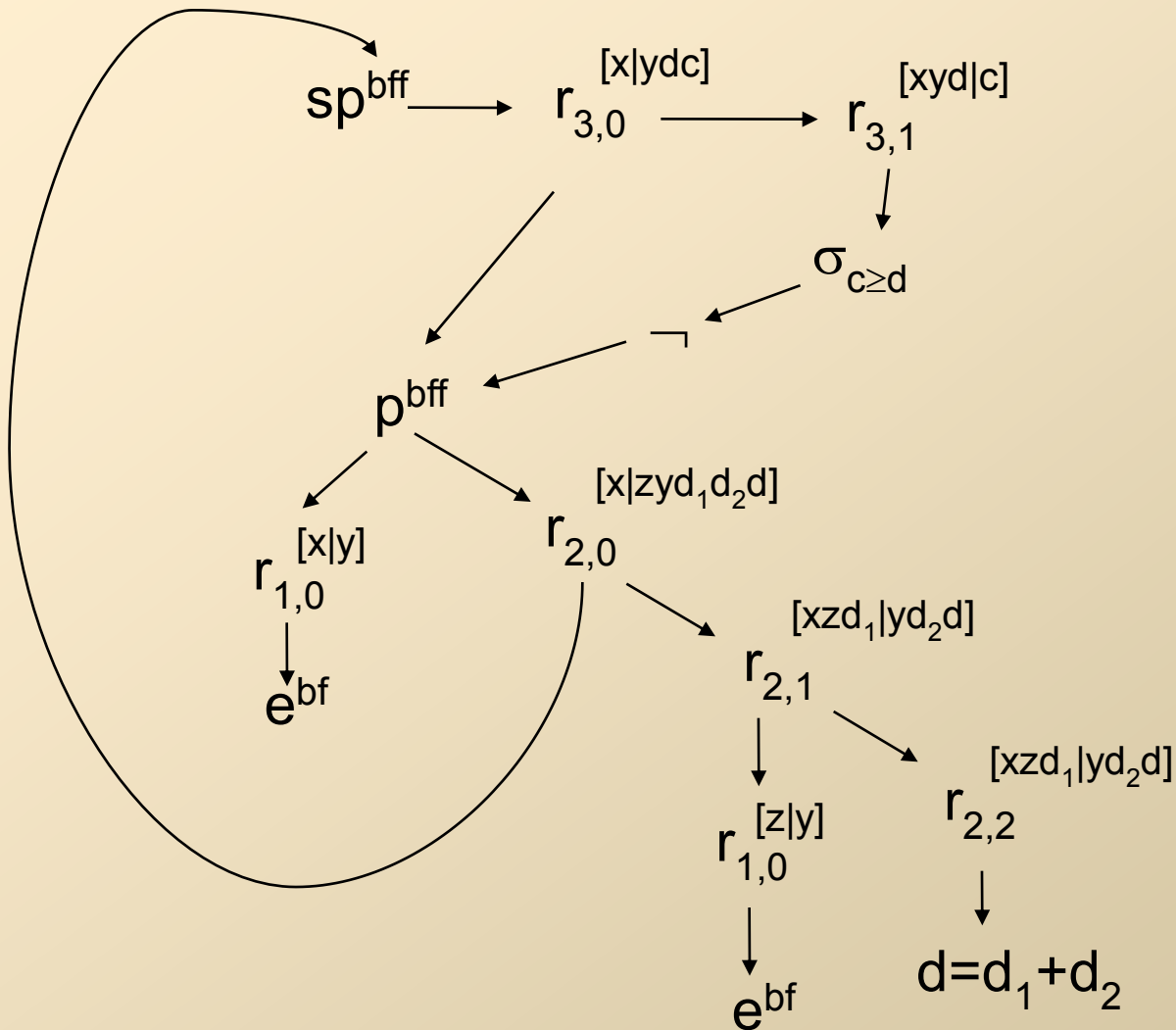
?- $sp(a,v,d)$

Optimalizácia pravidla r_3 :

$r_4: sp(x,y,d) \leftarrow p(x,y,d), \neg p(x,y, c \geq d).$

$r_5: sp(x,y,d) \leftarrow subtotal(p(x,y,d); x; d=\min(d)), p(x,y,d).$

Graf cieľov a pravidiel



Magické transformácie

Ján Šturc

Jar, 2014

Magická transformácia

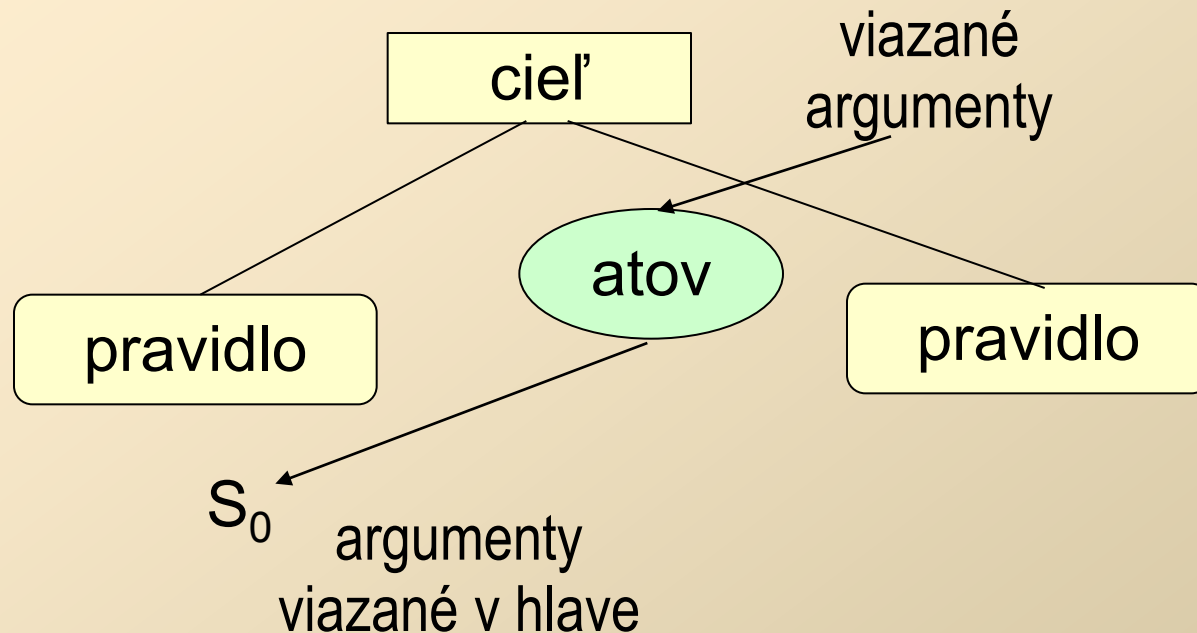
- Magická transformácia je optimalizačná technika, ktorá transformuje datalógový program na iný datalógový program s nasledujúcimi vlastnosťami:
 1. Počíta tú istú odpoveď ako pôvodný datalógový program. Nie je to celkom pravda, keď pôvodný program neskončí.
 2. Keď ho vypočítavame seminaívnou (naívnou) evaluáciou, do výpočtu vchádza tá istá množina faktov (n-tíc) ako pri výpočte zhora nadol (SLD-rezolúciou alebo RGT, či QRGT).
 3. Selekcie (väzby argumentov) sa posúvajú od cieľov k pocielom. Z hlavy pravidla postupne (z ľava do prava) k jednotlivým pocielom v tele.

Od realizovateľného RGG k „mágii“

- Nech je daný realizovateľný RGG
- Každému IDB predikátu p priradíme magický predikát m_p .
- Argumenty predikátu m_p zodpovedajú viazaným argumentom predikátu p . Predpokladáme, že p má uniformné ozdoby (unique binding pattern).
- m_p bude pravdivý práve pre tie n -tice, ktoré sa vyskytnú v niektorom uzle pre cieľ p pri „rozvíjaní“ zhora nadol.
- Pravidlovému uzlu r_{ij} priradíme pomocný predikát sup_{ij} .
- Argumentami sup_{ij} sú *viazané argumenty* pravidla („dosiahnuté“, v ozdobe pravidla sa nachádzajú [**tu** | ...].) a *zaujímavé* premenné na danom mieste pravidla.
- Premenná je zaujímavá na mieste j ak sa vyskytuje v hlave, alebo v podcieľoch napravo miesta j (s indexami väčšími ako j).

Vzory toku dát (widgets)

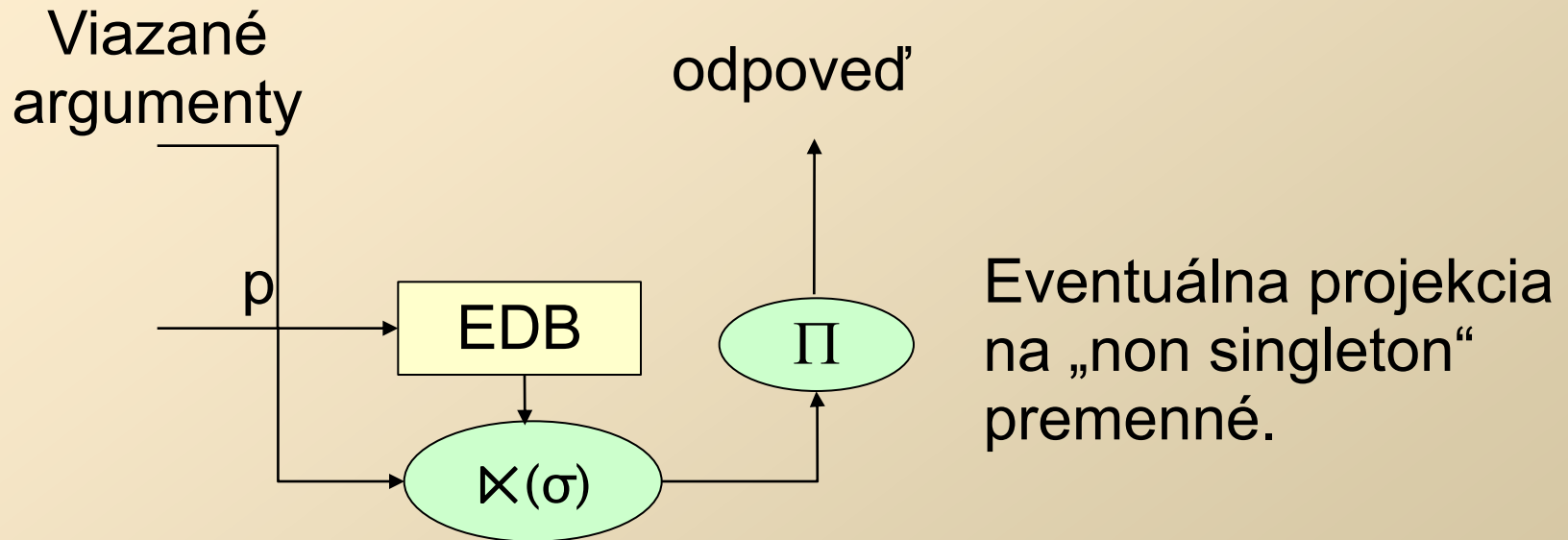
1. od cieľa k hlave pravidla



Odovzdanie viazaných argumentov cieľa hlave pravidla.

Vzory toku dát (widgets)

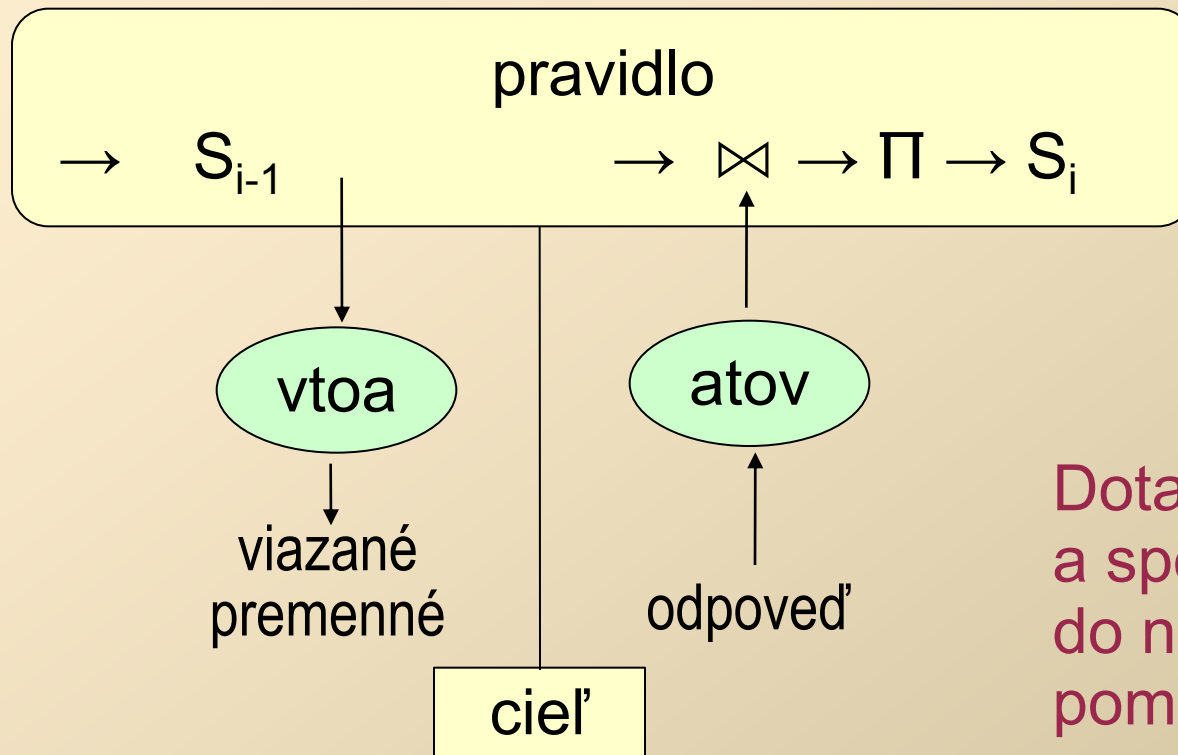
2. dotazy na EDB



Detail dotazu na EDB predikát.

Vzory toku dát (widgets)

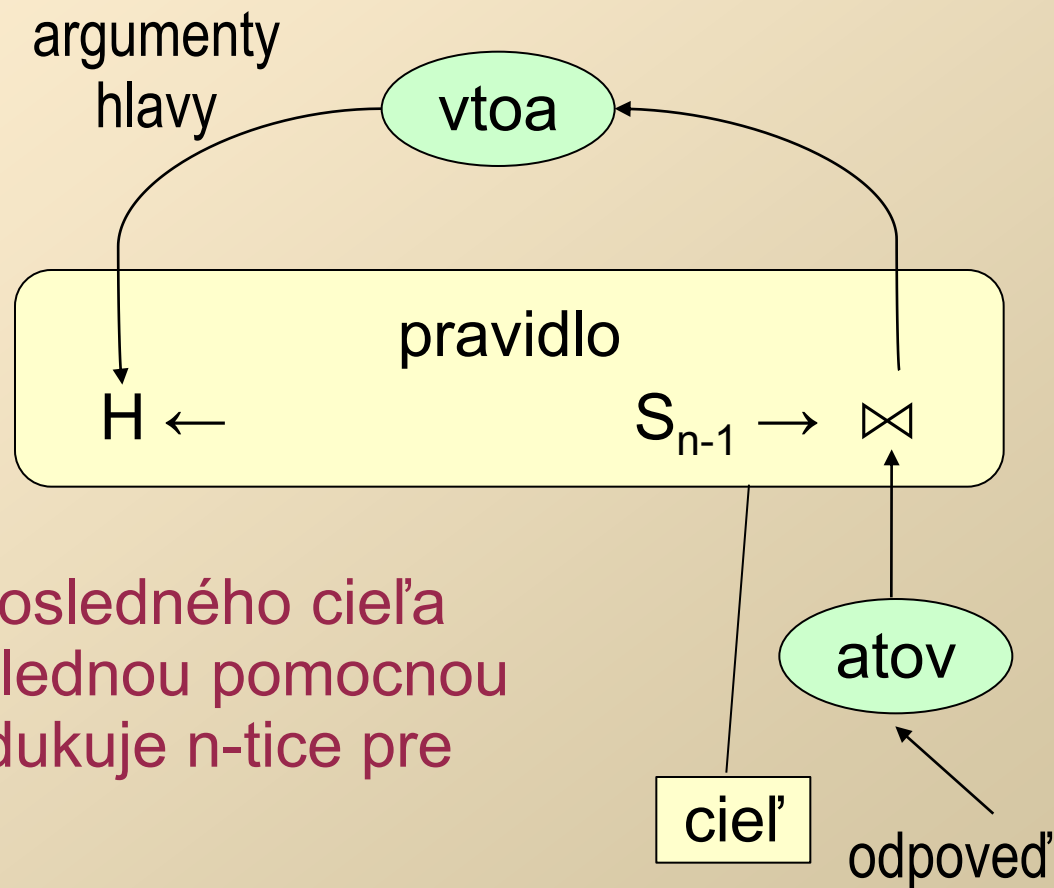
3. spracovanie v pravidle



Dotaz na podcieľ
a spojenie výsledkov
do nasledujúcej
pomocnej relácie.

Vzory toku dát (widgets)

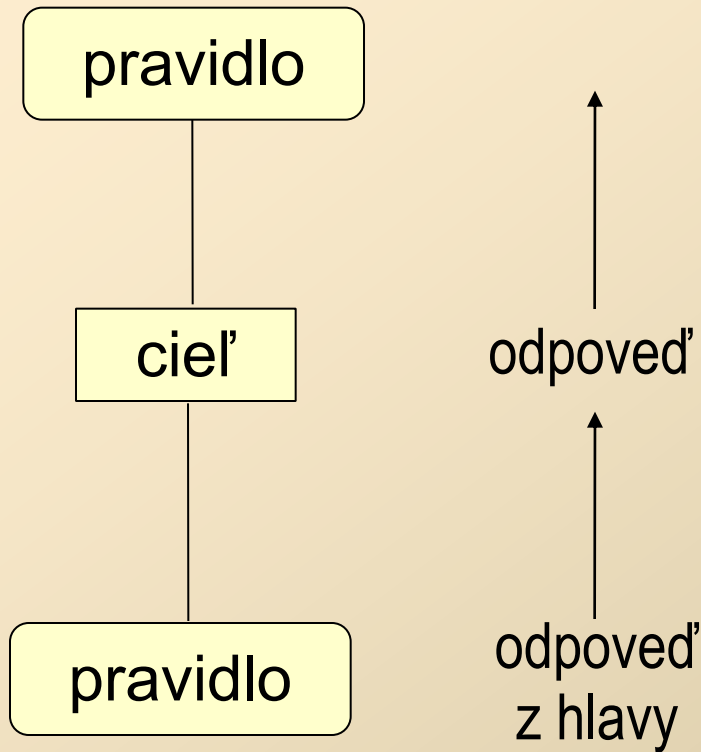
4. od posledného cieľa k hlave



Odpoveď od posledného cieľa sa spája s poslednou pomocnou reláciou a produkuje n-tice pre hlavu pravidla.

Vzory toku dát (widgets)

5. odpoved' od hlavy pravidla k cieľu



Odpovede sa posúvajú od hlavy pravidla syna k otcovi cieľu a od cieľa syna k pravidlu otcovi. Pokiaľ nedospejú do koreňa.

Päť skupín (groups) pravidiel pre magickú transformáciu

Pravidlá sú tvaru:

$$H \leftarrow G_1, G_2, \dots, G_n$$

Dotaz: $G(\text{viazané argumenty}, \text{voľné argumenty})$

Základnou myšlienkou magických transformácií je simulácia toku dát v grafe cieľov a pravidiel.

- Celý výpočet prebiehá v pomocných predikátoch
- Magické predikáty slúžia ako filtre, ktoré pri výpočte zdola nahor nepustia do výpočtu n-tice, ktoré nie sú v súlade s ozdobou v hlave (slide 5)
- Skupiny pravidiel 1 – 4 približne zodpovedajú vzorom toku dát
- Semeno sú viazané premenne v koreni (dotaze)

Magické transformácie 1

1. skupina

Pomocný predikát určuje magický pre nasledujúci podcieľ.

Ak G_i má IDB predikát p : $m_p := \text{atov}(G_i, \text{sup}_{r,i-1})$

$m_p(\text{viazané argumenty } G_i) \leftarrow \text{sup}_{r,i-1}(\text{premenné})$

2. skupina

Magický predikát určuje nultý pomocný.

Ak hlava pravidla má predikát q : $\text{sup}_{r,0} := \text{vtoa}(q, m_q)$

$\text{sup}_{r,0}(\text{premenné}) \leftarrow m_q(\text{viazané argumenty})$

3. skupina

$i-1$. pomocný predikát a G_i určujú i . Pomocný.

$\text{sup}_{r,i} := \text{sup}_{r,i-1} \bowtie G_i$

$\text{sup}_{r,i}(\text{premenné}) \leftarrow \text{sup}_{r,i-1}(\text{premenné}), G_i(\text{argumenty})$

Magické transformácie 2

4.skupina

Posledný pomocný predikát a posledný podcieľ určuje hlavu. $H := \text{atov}(H, \sup_{r,n-1} \bowtie G_n)$

$H(\text{argumenty}) \leftarrow \sup_{r,n-1}(\text{variables}), G_n(\text{argumenty})$

5. skupina

Inicializácia (semeno, seed). Ak dotaz je na predikát p , pridáme pravidlo:

$m_p(\text{viazané argumenty dotazu}).$

Je to jediné pravidlo, ktoré závisí na dotaze (the actual query). Ostatné pravidlá závisia len od pôvodných pravidiel (rules) a väzieb (binding patterns).

Možno názornejšie je

1. Zčať semenom (5. skupina)
2. Priradiť hranám postupujúcim od cieľa k pravidlu pravidlá pre nulté pomocné predikáty (2. skupina)
3. Priradiť vnútropravidlovým hranám postupové pravidlá $\text{sup}_{r,i} \leftarrow \text{sup}_{r,i-1}, G_i$ (3. skupina)
4. Spracovať konce pravidiel – posledné pravidlové uzly (3. skupina)
5. Nakoniec definovať magické predikáty pre hrany vedúce do intencionálnych predikátov (v rekurzívnych pravidlách sú to spätné hrany, môžu to byť aj dopredné hrany, ak pravidlo obsahuje „cudzí intencionálny predikát“).
 $\text{mag}_p \leftarrow \text{sup}...$ (1. skupina).

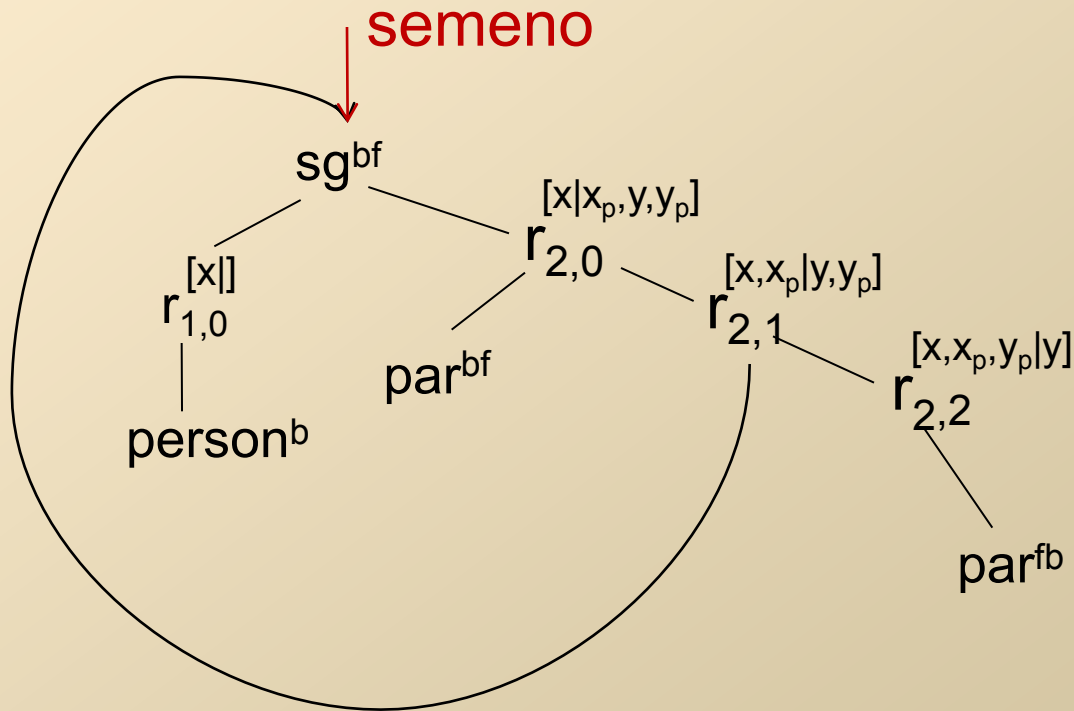
Pri správnom usporiadaní podcieľov a trochu praxe sa to dá robiť aj bez kreslenia RGG.

Príklad tá istá generácia RGG

$r_1: \text{sg}(x,x) \leftarrow \text{person}(x)$

$r_2: \text{sg}(x,y) \leftarrow \text{par}(x,x_p), \text{sg}(x_p,y_p), \text{par}(y,y_p)$

?- $\text{sg}(j,y)$



Príklad tá istá generácia „mágia“

1. skupina

$m_sg(x_p) \leftarrow sup_{2,1}(x, x_p)$

2. skupina

$sup_{1,0}(x) \leftarrow m_sg(x)$

$sup_{2,0}(x) \leftarrow m_sg(x)$

3. skupina

$sup_{2,1}(x, x_p) \leftarrow sup_{2,0}(x), par(x, x_p)$

$sup_{2,2}(x, y_p) \leftarrow sup_{2,1}(x, x_p), sg(y, y_p)$

4. skupina

$sg(x, x) \leftarrow sup_{1,0}(x), person(x)$

$sg(x, y) \leftarrow sup_{2,2}(x, y_p), par(y, y_p)$

5. semeno

$m_sg(j).$

Inak

Dôležité sú nulté pomocné predikáty, ktoré filtrujú výpočty pravidiel'.

$r_1: sg(x,x) \leftarrow sup_0(x), person(x)$

$r_2: sg(x,y) \leftarrow sup_0(x), par(x,x_p), sg(x_p,y_p), par(y,y_p)$

$r_3: sup_0(x) \leftarrow x=j$

$r_4: sup_0(x) \leftarrow sup_0(z), par(z,x)$

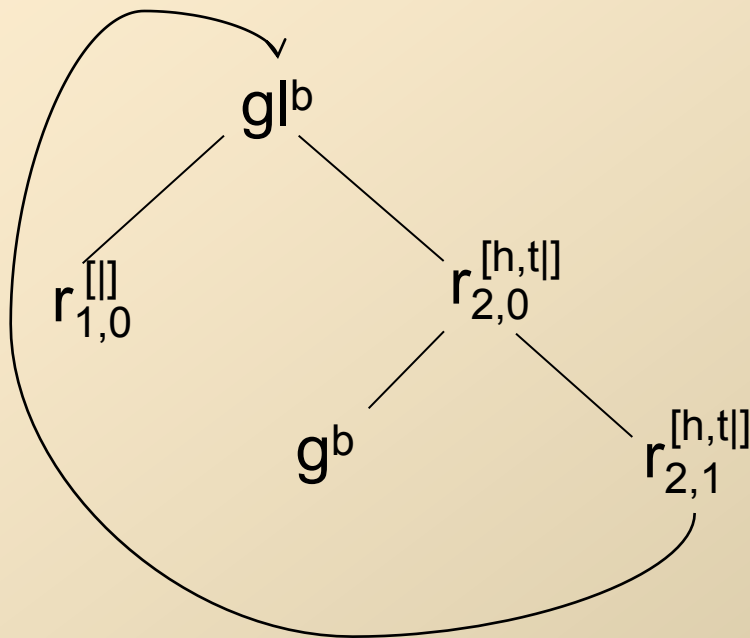
Kúzla, čary, mágia? Nie! Len jednoduchá optimalizácia pomocou filtra sup_0 , ktorý riadi, ktoré hodnoty x pustíme do výpočtu.

Príklad „good list“

r_1 : $gl(nil)$

r_2 : $gl(cons(h,t)) \leftarrow g(h), gl(t).$

?- $gl(l_0)$



1. skupina

$m_gl(t) \leftarrow sup_{2,1}(h,t).$

2. skupina

$sup_{1,0} \leftarrow m_gl(nil)$

$sup_{2,0}(h,t) \leftarrow m_gl(cons(h,t)).$

3. skupina

$sup_{2,1}(h,t) \leftarrow sup_{2,0}(h,t), g(h).$

4. skupina

$gl(nil) \leftarrow sup_{1,0}$

$gl(cons(h,t)) \leftarrow sup_{2,1}(h,t), gl(t).$

5. semeno

$m_gl(l_0).$

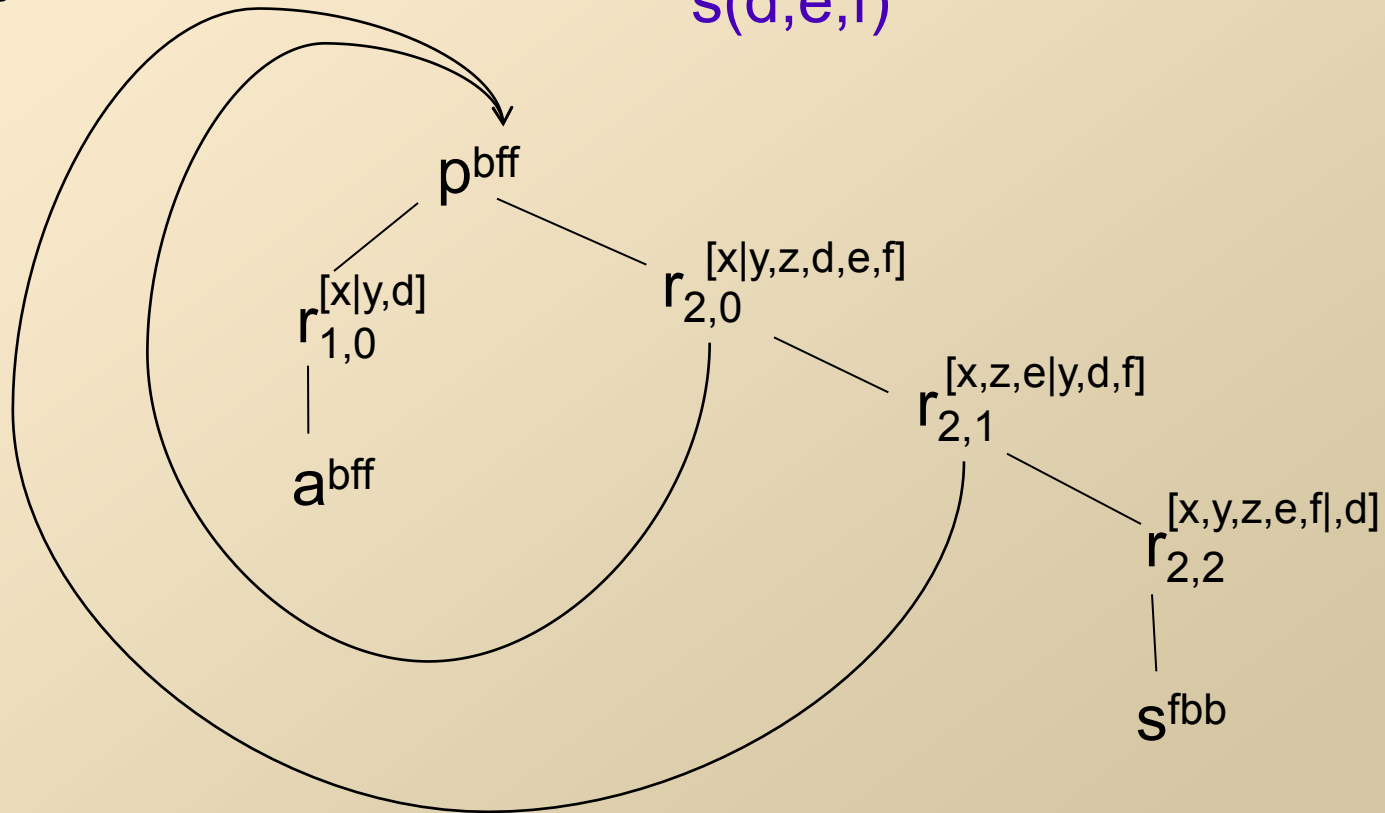
Príklad – ohodnotené cesty v grafe

r1: $p(x,y,d) \leftarrow a(x,y,d)$

r2: $p(x,y,d) \leftarrow p(x,z,e), p(z,y,f), d=e+f$

?- $p(x_0,y,d)$

$s(d,e,f)$



Ohodnotené cesty v grafe - „mágia“

1. skupina

$$m_p(x) \leftarrow \sup_{2,0}(x)$$

$$m_p(z) \leftarrow \sup_{2,1}(x,z,e)$$

2. skupina

$$\sup_{1,0}(x) \leftarrow m_p(x)$$

$$\sup_{2,0}(x) \leftarrow m_p(x)$$

3. skupina

$$\sup_{2,1}(x,z,e) \leftarrow \sup_{2,0}(x), p(x,z, e)$$

$$\sup_{2,2}(x,y,e,f) \leftarrow \sup_{2,1}(x,z,e), p(z,y,f)$$

4. skupina

$$p(x,y,d) \leftarrow \sup_{1,0}(x), a(x,y,d)$$

$$p(x,y,d) \leftarrow \sup_{2,2}(x,y,e,f), d=e+f$$

5. semeno: $m_p(x_0)$

Nerekurzívny príklad v SQL.

```
select Z1.meno  
from Zamestnanci Z1  
where Z1.funkcia = 'IT špecialista' and  
Z1.plat < ( select AVG(Z2.plat)  
from Zamestnanci Z2  
WHERE Z2.oddelenie = Z1.oddelenie);
```

Prepis do datalógu:

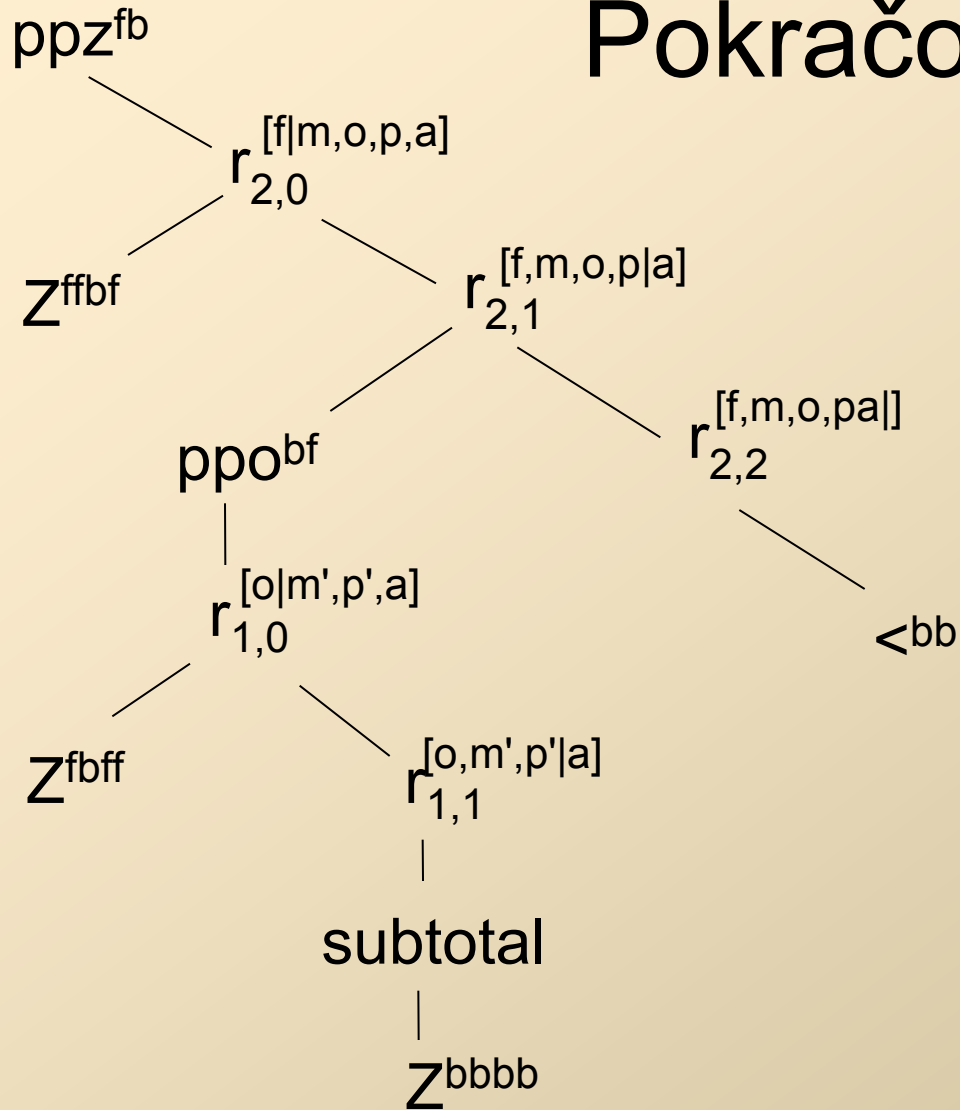
EDB: $Z(m,o,f,p)$

$r_1: \text{ppo}(o,a) \leftarrow \text{subtotal}(Z(m,o,f,p); o; a=\text{avg}(\text{plat}))$

$r_2: \text{ppz}(m,f) \leftarrow Z(m,o,f,p), \text{ppo}(o,a), p < a$

?- $\text{ppz}(m, \text{'IT'})$

Pokračovanie RGG



Pokračovanie – magický program

1. skupina

$m_ppo(o) \leftarrow sub_{2,1}(m,f,o,p)$

2. skupina

$sub_{1,0}(o) \leftarrow m_ppo(o)$

$sub_{2,0}(f) \leftarrow m_ppz(f)$

3. skupina

$sub_{1,1}(m',o,p') \leftarrow sub_{1,0}(o), Z(m',f',o,p')$

$sub_{2,1}(m,f,o,p) \leftarrow sub_{2,0}(o), Z(m,f,o,p)$

$sub_{2,2}(m,f,p,a) \leftarrow sub_{2,1}(m,f,o,p), ppo(o,a)$

4. skupina

$ppo(o,a) \leftarrow subtotal(sub_{1,1}(m',o,p'); o; a = avg(p'))$

$ppz(m,f) \leftarrow sub_{2,2}(m,f,p,a), p < a$

5. semeno: $m_ppz('IT')$.

Zovšeobecnená magická transformácia

V niektorých prípadoch je nevhodné obmedziť magické predikáty len na viazané argumenty cieľov. Informácia v ostatných premenných sa dá pri výpočte využiť.

Ponecháme preto magickým predikátom tie isté argumenty ako majú príslušné IDB predikáty.

Tým sa zmenia pravidlá prvej a druhej skupiny. V ostatných skupinách pribudnú potrebné premenné. Semeno sa zmení tak, že obsahuje všetky argumenty dotazu.

- 1. skupina: $m_p(t_1, \dots, t_n) \leftarrow \sup_{j,i-1}(x_1, \dots, x_m)$
- 2. skupina: $\sup_{j,0}(x_1, \dots, x_m) \leftarrow m_p(t_1, \dots, t_n)$
- 5. semeno: $m_p(t_1, \dots, t_n)$

Vypočet zovšeobecnených magických programov

- Semeno obsahuje neuzavreté termy pre voľné argumenty. Programy a dáta môžu obsahovať termy
- Algebraický stroj (db-engine) vyžaduje miernú modifikáciu. Joiny a antijoiny sa robia podľa unifikácie a nie podľa rovnosti.
- Odstránenie duplikátov sa robí na základe subsumcie. Možno je dobré „duplikáty“ vôbec nevyrábať.
 - Vyhľadanie podľa rovnosti dokážeme robiť pomocou B-stromov, či hašovania.
 - Zovšeobecnenie pre unifikáciu a subsumciu je netriviálne. Asi všetky programy budú v najhoršom prípade kvadratické (nested loops).

Relation algebra revisited I.

- Storing relations
 - We tacitly assume that EDB relations contains ground terms only.
 - The IDB relations consists of three sections:
 - **Variable** a boolean value indicating if the relation contains a tuple consisting of mutually distinct variables.
 - **Non-ground terms**, where all tuples containing a non-ground term are stored. The names of variables in the particular tuple are considerable.
 - **Rest**, this is a ordinary relation containing only ground terms.
 - **Duplicates** are tuples subsumed by some other tuple.
 - Maybe, a good idea is to use **special function symbols** f^n , which bind n arguments and to deal with unary relations only.

Relation algebra revisited II.

- Projection f^k of the tuple f^n on its $k \leq n$ arguments i_1, \dots, i_k .
$$f^k(X_{i_1}, \dots, X_{i_k}) \leftarrow f^n(X_1, \dots, X_n).$$

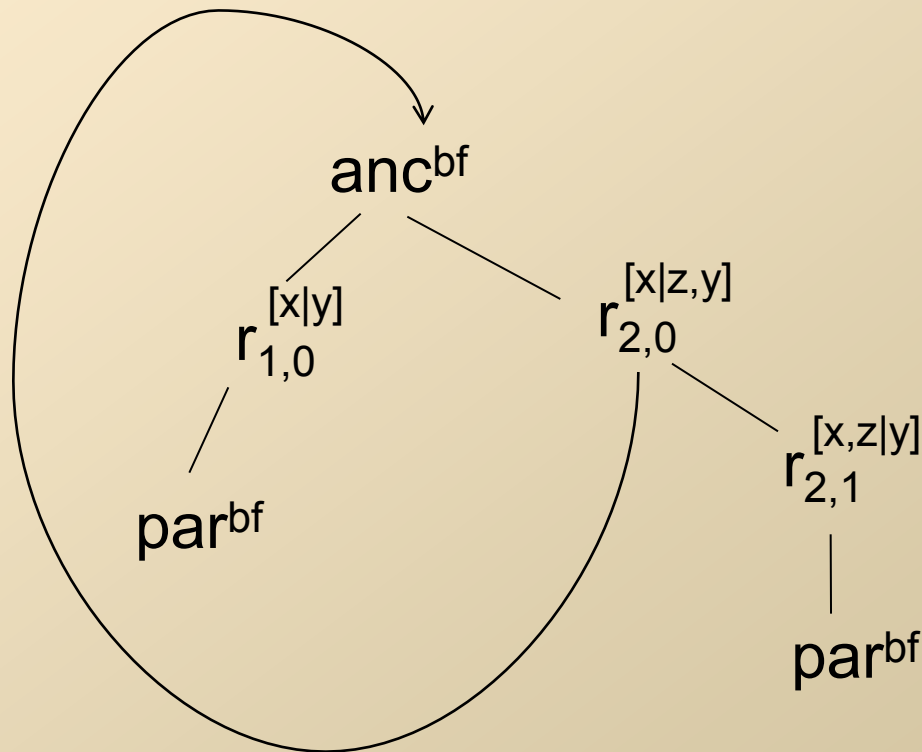
Note, that this “rule” define only one tuple.
- Join and semijoin of two tuples $f^m(X_1, \dots, X_m)$ and $f^n(Y_1, \dots, Y_n)$ on $i_1=j_1, \dots, i_k=j_k$.
 1. Compute projections $f^k(X_{i_1}, \dots, X_{i_k})$ and $f^k(Y_{i_1}, \dots, Y_{i_k})$.
 2. Unify these tuples as terms using weak unification. Let τ be mgu if exists.
 3. If the tuples do not unify their join and semijoin is empty set. Else $f^m \bowtie f^n = \{f^{m+n-k}(X_1, \dots, X_m, Y) \circ \tau\}$ and $f^m \ltimes f^n = \{f^{m+n-k}(X_1, \dots, X_m) \circ \tau\}$, where Y are arguments of the second tuple except selected k ones.

Zjednodušenia a drobné optimalizácie

- Pravidlá druhej skupiny nepotrebujeme. Namiesto nultých pomocných predikátov môžeme rovno použiť magické predikáty.
- Pravidlá, kde ľavá strana subsumuje pravú možno vynechať.
- Vo vnútri postupnosti po sebe idúcich EDB predikátov sú pomocné predikáty zbytočné (na celú konjunkciu sa môžeme dívať ako na jeden EDB predikát) a netreba ich generovať (t.j. ani pravidlá pre nich).
- Ak sa pomocný predikát nachádza pred EDB predikátom, vyskytuje sa iba raz, môžeme ho eliminovať (vynechať pravidlo, kde je na ľavej strane a jeho výskyt nahradiť pravou stranou tohto pravidla).

Príklad – predkovia Johna

$r_1: \text{anc}(x,y) \leftarrow \text{par}(x,y)$ /* $\text{par}(x,y)$: y je rodič x , anc – predok */
 $r_2: \text{anc}(x,y) \leftarrow \text{anc}(x,z), \text{par}(z,y)$
?- $\text{anc}('j', y)$



Predkovia Johna – magická a zjednodušená magická transformácia

(1) ~~$m_anc(x) \leftarrow sup_{2.0}(x)$~~
(2) ~~$sup_{1.0}(x) \leftarrow m_anc(x)$~~
 ~~$sup_{2.0}(x) \leftarrow m_anc(x)$~~ $m_anc(x)$
(3) ~~$sup_{2.1}(x,z) \leftarrow sup_{2.0}(x), anc(x,z)$~~
(4) ~~$anc(x,y) \leftarrow sup_{1.0}(x), par(x,y)$~~
 ~~$anc(x,y) \leftarrow sup_{2.1}(x,z), par(z,y)$~~
(5) $m_anc('j')$.

Po zjednodušení:

$anc(x,y) \leftarrow m_anc(x), par(x,y).$
 $anc(x,y) \leftarrow m_anc(x), anc(x,z), par(z,y).$
 $m_anc('j').$

Príklad – syntaktická analýza

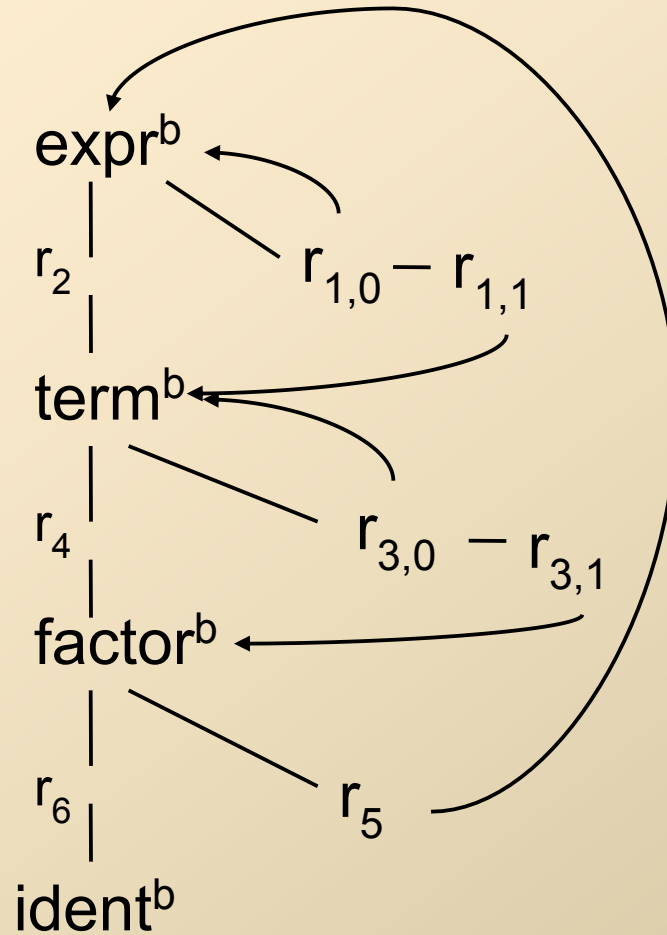
Pravidlá:

r_1 : $\text{expr}(\text{plus}(E,T)) \leftarrow \text{expr}(E), \text{term}(T).$
 r_2 : $\text{expr}(T) \leftarrow \text{term}(T).$
 r_3 : $\text{term}(\text{mult}(T,F)) \leftarrow \text{term}(T), \text{factor}(F).$
 r_4 : $\text{term}(F) \leftarrow \text{factor}(F).$
 r_5 : $\text{factor}(\text{bra}(E)) \leftarrow \text{expr}(E).$
 r_6 : $\text{factor}(F) \leftarrow \text{ident}(F).$

Cieľ: expr^b *rozpoznanie konkrétneho výrazu e .*

Príklad – syntaktická analýza

(graf cieľov a pravidiel)



Pretože expr^b znamená, že dekomponujeme známy výraz. Všetky zaujímavé premenné v pravidlách sú viazané. Ozdoba párnych pravidiel je $[x]$ a ozdoba pravidlových uzlov s nepárny prvým indexom $[x, y]$.

Príklad – syntaktická analýza (magické pravidlá)

1. skupina:

$m_expr(E) \leftarrow sup_{1,0}(E,T).$
 $m_expr(E) \leftarrow sup_5(E).$
 $m_term(T) \leftarrow sup_2(T).$
 $m_term(T) \leftarrow sup_{1,1}(E,T).$
 $m_term(T) \leftarrow sup_{3,0}(T,F).$
 $m_factor(F) \leftarrow sup_4(F).$
 $m_factor(F) \leftarrow sup_{3,1}(T,F).$
 $m_ident(I) \leftarrow sup_6(I).$

2. skupina:

$sup_{1,0}(E,T) \leftarrow m_expr(plus(E,T)).$
 $sup_{3,0}(T,F) \leftarrow m_term(mult(T,F)).$
 $sup_5(E) \leftarrow m_factor(bra(E)).$
 $sup_2(T) \leftarrow m_expr(T).$
 $sup_4(F) \leftarrow m_term(F).$
 $sup_6(I) \leftarrow m_factor(I).$

Príklad – syntaktická analýza

(magické pravidlá – pokračovanie)

3. skupina: $\text{sup}_{1,1}(E,T) \leftarrow \text{sup}_{1,0}(E,T), \text{expr}(E).$
 $\text{sup}_{3,1}(T,F) \leftarrow \text{sup}_{3,0}(T,F), \text{term}(T).$

4. skupina: $\text{expr}(E) \leftarrow \text{sup}_{1,1}(E,T), \text{term}(T).$
 $\text{expr}(E) \leftarrow \text{sup}_2(E), \text{term}(E).$
 $\text{term}(T) \leftarrow \text{sup}_{3,1}(T,F), \text{factor}(F).$
 $\text{term}(T) \leftarrow \text{sup}_4(T), \text{factor}(T).$
 $\text{factor}(F) \leftarrow \text{sup}_5(F), \text{expr}(F).$
 $\text{factor}(F) \leftarrow \text{sup}_6(F), \text{ident}(F).$

5. semeno: $\text{m_expr}(\mathbf{e}).$ Pozn.: \mathbf{e} je zadaný výraz.

Príklad – syntaktická analýza (zovšeobecnená mágia)

m_expr(E) ← m_expr(plus(E,T)).
m_expr(E) ← m_factor(bra(E)).
m_term(T) ← m_expr(T).
m_term(T) ← m_expr(plus(E,T)), expr(E).
m_term(T) ← m_term(mult(T,F)).
m_factor(F) ← m_term(F).
m_factor(F) ← m_term(mult(T,F)), term(T).
expr(E) ← m_expr(plus(E,T)), expr(E), term(T).
expr(E) ← m_expr(E), term(E).
term(T) ← m_term(mult(T,F)), term(T), factor(F).
term(T) ← m_term(T), factor(T).
factor(F) ← m_factor(bra(F)), expr(F).
factor(F) ← m_factor(F), ident(F).
m_expr(e).

Príklad – syntaktická analýza

(O čom to vlastne je ?)

- Magické predikáty zariadia, že do výpočtu môžu vstupovať len podvýrazy zadaného výrazu **e**.
- Jediný extenzionálny predikát `ident` slúži na kontrolu, či identifikátor bol deklarovaný.
- Oproti programu syntaktickej analýzy je tu len malá neefektívnosť v tom, že do `m_term` a `m_factor` sa dostávajú aj podvýrazy **e**, ktoré nie sú termy resp. faktory.
- Všimnite si, že netransformovaný program a program pre `exprf` sa môže eventuálne zacykliť. Vyžaduje dodatočnú inteligenciu na to, aby v konečnom čase vygeneroval všetky výrazy kratšie než zadaná dĺžka.

Špecializácia rektifikáciou

- Niekedy netreba mágia stačí dôsledná rektifikácia programu včítane dotazu.

r_1 : $\text{anc}(X, Y) \leftarrow \text{par}(X, Y)$ /* $\text{par}(x, y)$: y je rodič x , anc – predok*/

r_2 : $\text{anc}(X, Y) \leftarrow \text{anc}(X, Z), \text{par}(Z, Y)$

?- $\text{anc}('j', Y)$

r_3 : $\text{ancj}(Y) \leftarrow \text{par}(j, Y).$

r_4 : $\text{ancj}(Y) \leftarrow \text{ancj}(Z), \text{par}(Z, Y)$

?- $\text{ancj}(Y)$

- Zistite, ktoré z programov tejto prednášky sa dajú optimalizovať rektifikáciou!

Ako je to s negáciou ?

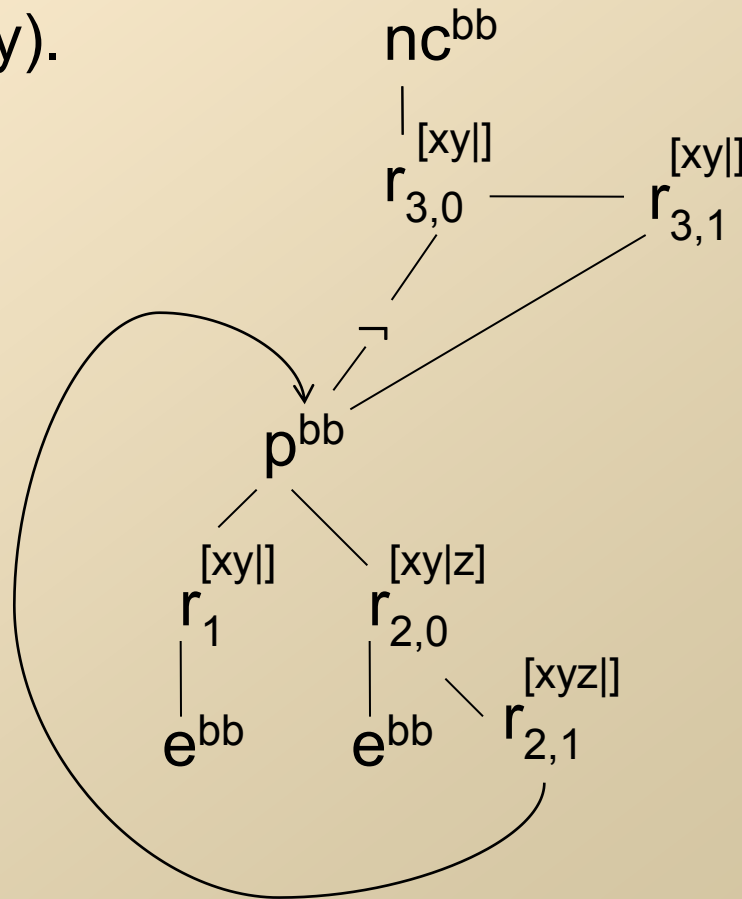
$r_1: p(x,y) \leftarrow e(x,y).$

$r_2: p(x,y) \leftarrow e(x,z), p(z,y).$

$r_3: nc(x,y) \leftarrow \neg p(y,x), p(x,y).$

?- nc(a,d)

stratifikovaný program



Po magickej transformácii

$p^{bb}(x, y) \leftarrow m_p^{bb}(x, y), e(x, y).$

$p^{bb}(x, y) \leftarrow m_p^{bb}(x, y), e(x, z), p^{bb}(z, y).$

$nc^{bb}(x, y) \leftarrow m_nc^{bb}(x, y), \neg p^{bb}(y, x), p^{bb}(x, y).$

$m_p^{bb}(y, x) \leftarrow m_nc^{bb}(x, y).$

$m_p^{bb}(x, y) \leftarrow m_nc^{bb}(x, y), \neg p^{bb}(y, x).$

$m_p^{bb}(z, y) \leftarrow m_p^{bb}(x, y), e(x, z).$

$m_nc(a, d).$

Program nie je stratifikovaný: $p^{bb} \rightarrow m_p^{bb} \rightarrow \neg p^{bb}.$

Všimnite si, že ak dodržíte pravidlo, písať negované predikáty až po všetkých pozitívnych predikátoch, sa môže podariť konštrukcia programu, kde magická transformácia nezachováva stratifikovanosť, iba ak program obsahuje pravidlo s aspoň dvomi negovanými predikátmi.

Mágia a „well-founded“ sémantika

$p(a) \leftarrow q(a), \neg r(a).$

$q(a) \leftarrow \neg p(a).$

$\text{EDB} = \emptyset.$

$r(a).$

wf. model $M = \{r(a), q(a), \neg p(a)\}$

? - $q(a)$

Po magickej transformácii:

$p(a) \leftarrow m_p(a), q(a), \neg r(a).$

$q(a) \leftarrow m_q(a), \neg p(a).$

$r(a) \leftarrow m_r(a).$

$m_p(a) \leftarrow m_q(a).$

$m_q(a) \leftarrow m_p(a).$

$m_r(a) \leftarrow m_p(a), q(a).$

$m_q(a).$

wf. model $M' = \{m_q(a), m_p(a)\}$

Nezachováva ani M , ani správnu odpoveď na dotaz.

Diskusia

- Príklad je značne umelý
 - Použitie magickej transformácie na program, ktorý neobsahuje premenné.
 - Fakt $r(a)$ je postulovaný v programe namiesto, aby bol uložený v EDB.
- Naznačuje to, že pri použití magickej transformácie na programy s negáciou, musíme byť opatrní a najprv zhodnotiť výsledok, či je korektný.
- Možný jej aj taký prístup, že vyšpecifikujeme dostatočne širokú podtriedu datalógových programov, pre ktoré magickou transformáciou nedochádza k zmene sémantiky, alebo aspoň odpoveď sa zachová.

Literatúra

S. Abiteboul, R. Hull, V. Vianu: *Foundations of Databases*. Addison Wesley, Reading MA, 1995

J. D. Ullman: *Database and Knowledge-Base Systems*, Vol. II. Computer Science Press, New York, 1988

Website www-db.stanford.edu/~ullman/cs345-notes.html

C. Beeri, R. Ramakrishnan: *On the Power of magic*. In PODS, 1987.

D. B. Kemp, D. Srivastava, P. J. Stuckey: Bottom-up Evaluation and Query Optimization of Well-Founded Models. Theoretical Computer Science (1995) .

Niektoré projekty

Name	Evaluation	Syntactic Restrictions	Negation	Data Requirements
CORAL (Wisconsin)	Magic Templates	First-order	Modularly Stratified	Main Memory
LDL (Texas)	Magic Sets	First-order	Modularly Stratified	Main Memory
Glue-Nail (Stanford)	Magic Rewriting	Restricted HiLog Extensions	Well-founded	Main Memory
XSB	SLG	HiLog with F-O Optimizations	Well-founded	Main Memory

Optimalizácie na úrovni relačnej algebry

Ján Šturc

Jar, 2013

Formy reprezentácie výrazov

- Sú to algebraické výrazy klasická reprezentácia
 - stromy
 - syntaktický strom
 - strom výrazu
 - dagy (orientované acyklické grafy)
 - spoločné podvýrazy sa vyskytujú iba raz
- Pri relačnej algebre (videli sme to aj v datológových optimalizáciach) nás veľmi často zaujímajú spoločné premenné (atribúty) výrazov a podvýrazov. Vhodná reprezentácia pre túto informáciu je:
 - hypergraf

Hypergrafy (hypergraphs)

Definícia:

Nech X je neprázdna množina a $\mathcal{E} = \{E_i\}_{i=0}^{m-1}$ je systém neprázdnych podmnožín X takých, že $X = \bigcup_{i=0}^{m-1} E_i$.

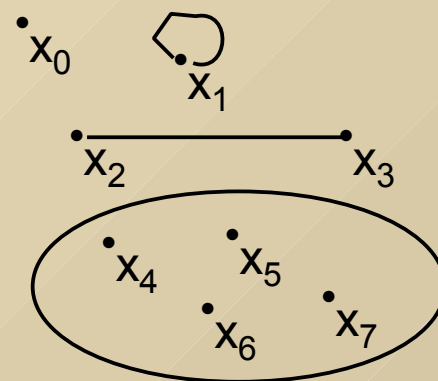
Dvojicu $\langle X, \mathcal{E} \rangle$ nazývame hypergrafom.

Prvky množiny X nazývame vrcholy (uzly) a množiny $E_0 \dots E_{m-1}$ nazýme **hyperhrany**.

Ak pre všetky $i < m$, je $|E_i| = 2$. Je to obyčajný graf.

Kreslenie hypergrafov:

- Vrchol znázorňujeme bodom.
- Hranu kardinality 1, slučkou.
- Hranu kardinality 2, úsečkou.
- Všetky ostatné hrany oválom.



Často všetky hrany oválom.

Prípadne inou uzavretou čiarou.

Niektoré vlastnosti hypergrafov

- Úlohy, ktoré riešime pre grafy, sú často zaujímavé aj pre hypergrafy (súvislosť, cykly, kliky, farbenie).
- **Incidenčná matica** nech $X = \{x_0, \dots, x_{n-1}\}$. Ku každému hypergrafu môžeme priradiť booleovskú maticu A typu $m \times n$ takú, že $a_{ij} = 1$ práve vtedy, keď $x_j \in E_i$.
- Naopak, každej booleovskej matici, ktorá má aspoň jednu jednotku v každom riadku a v každom stĺpci zodpovedá nejaký hypergraf.
- Nech H je hypergraf a s maticou A , potom hypergraf H^* , ktorý zodpovedá A^T sa nazýva **duálnym** k hypergrafu H .
- Zrejme $H^{**} = H$.

GYO redukcia hypergrafu (Graham, Yu a Ozsoyoglu)

- GYO redukcia je algoritmus postupnej dekompozície hypergrafu.

Pokiaľ sa niečo dá odstrániť opakuj kroky:

1. Ak nejaká hrana $E_i \subseteq E_j$, potom odstráň E_i .
2. Ak pre nejaký vrchol $x \in E$ neexistuje hrana $E' \neq E$ taká, že $x \in E'$, potom odstráň vrchol x .

Podľa definície hypergraf nemá prázdne hrany. V dôsledku toho pravidlo (2) likviduje izolované hrany.

GYO redukt (výsledok redukcie) je určený jednoznačne a postup má Church - Roserovú vlastnosť. (Nezáleží na tom, ktorý z možných krokov si vyberieme.)

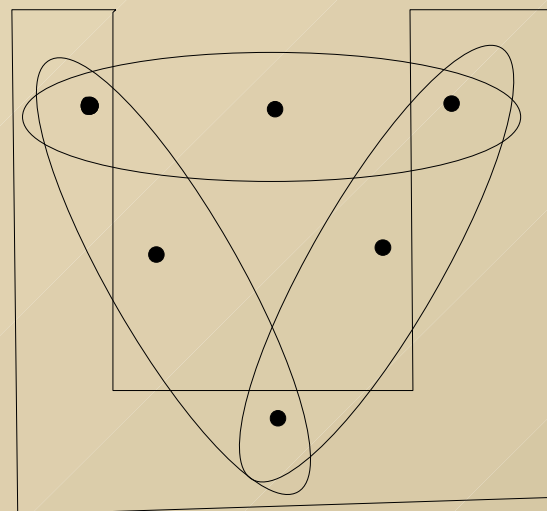
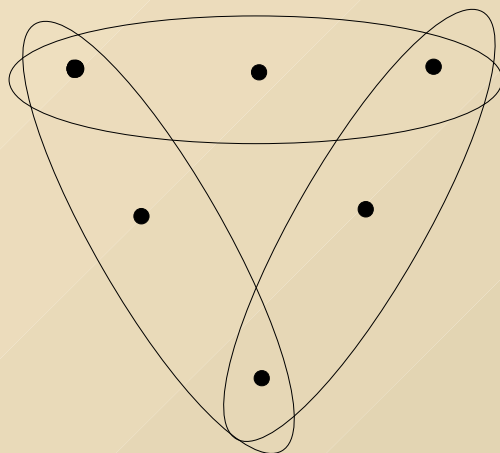
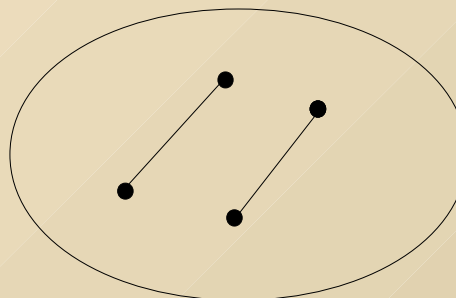
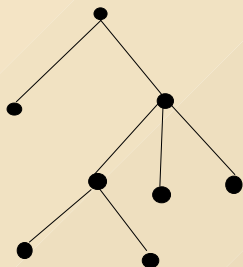
Hypergraf sa nazýva GYO redukovateľný, ak následkom GYO redukcie „zmizne“. (Výsledok je prázdna hrana.)

V databázovej teórii sú to **acyklické hypergrafy**.

Trhanie uší

- Databázová podoba GYO redukcie
- Hyperhrana E hypergrafu \mathcal{H} je ucho
 1. Ak E je izolovaná hyperhrana
 2. Ak existuje hyperhrana F (svedok uchovitosti) taká, že pre každý vrchol $N \in E - F$ platí, že nie je incidentný so žiadnou inou hyperhranou hypergrafu \mathcal{H} okrem E .
- Algoritmus redukcie spočíva v postupnom odstraňovaní uší.
 - Hovoríme, že odstraňujeme ucho (hyperhranu) E v prospech svedka uchovitosti F .

Príklady



Algebra tabuliek versus algebra relácií

- Matematika
 - n-tica funkcia ***z usporiadanej*** množiny $\{1, \dots, n\}$ do množiny argumentov (hodnôt).
 - Pri skladní n-tíc $\langle a, b \rangle, c \rangle$ nie je to isté ako $\langle a, \langle b, c \rangle \rangle$.
- Databázy
 - n-tica (tuple) je funkcia z množiny mien (atribútov) ***neusporiadanej*** do množiny argumentov (hodnôt).
 - Pri skladní (join) n-tíc ich domény (obory definície) zjednocujeme.
 - Potreba technickej operácie premenovania (Π).
- Databázová relačná algebra nie je algebra relácií v matematike.

Zákony relačnej algebry 1

1. Join je komutatívny asociatívny a idempotentný

- $R \bowtie S \equiv S \bowtie R$
- $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
- $R \bowtie R \equiv R$

2. Tie isté pravidlá platia pre zjednotenie

- $R \cup S \equiv S \cup R$
- $(R \cup S) \cup T \equiv R \cup (S \cup T)$
- $R \cup R \equiv R$

3. Kartézsky súčin je join (vyžaduje disjunktné atribúty)

- $R \times S \equiv S \times R$ (Nehovorte matematikom !)
- $(R \times S) \times T \equiv R \times (S \times T)$
- $R \times R$ (Kartézska mocnina, vyžaduje explicitné premenovanie atribútov)

Zákony relačnej algebry 2

4. Selekcie sú joiny (presnejšie prieniky s nekonečnými reláciami)

- $\sigma_F(\sigma_G R) \equiv \sigma_G(\sigma_F R) \equiv \sigma_{F \wedge G} R$
- Ak $F \Rightarrow G$, potom $\sigma_F(\sigma_G R) \equiv \sigma_F R$

5. Projekcia

- Nech A a B sú množiny atribútov (premenných) a $A \subseteq B$, potom $\Pi_A(\Pi_B R) \equiv \Pi_A R$.

6. Selekcie a projekcie

- Nech A je množina atribútov F podmienka. Označme $B = A \cup \text{atrib}(F)$. Potom $\Pi_A(\sigma_F R) \equiv \Pi_A(\sigma_F(\Pi_B R))$

7. Projekcia a zjednotenie

- $\Pi_A(R \cup S) \equiv \Pi_A R \cup \Pi_A S$

Zákony relačnej algebry 3

8. Distributívne zákony

- $R \bowtie (S \cup T) \equiv (R \bowtie S) \cup (R \bowtie T)$
- $R \bowtie (S - T) \equiv (R \bowtie S) - (R \bowtie T)$
- $R \bowtie (S \ltimes T) \equiv (R \bowtie S) \ltimes (R \bowtie T)$
- $R \times (S \cup T) \equiv (R \times S) \cup R \times T$
- $R \times (S - T) \equiv R \times S - R \times T$
- $R \times (S \ltimes T) \equiv R \times S \ltimes R \times T$
- $\sigma_F(R \cup S) \equiv \sigma_F R \cup \sigma_F S$
- $\sigma_F(R - S) \equiv \sigma_F R - \sigma_F S$
- $\sigma_F(R \ltimes S) \equiv \sigma_F R \ltimes \sigma_F S$

Zákony relačnej algebry 4

9. Projekcia a join: Nech R a S sú relačné výrazy. Nech $A \subseteq (\text{atrib}(R) \cup \text{atrib}(S))$, nech $B = (A \cup \text{atrib}(S)) \cap \text{atrib}(R)$ a $C = (A \cup \text{atrib}(R)) \cap \text{atrib}(S)$. Potom
- $\Pi_A(R \bowtie S) \equiv \Pi_A(\Pi_B R \bowtie \Pi_C S)$
 - $\Pi_A(R \times S) \equiv \Pi_B R \times \Pi_C S$
10. Kombinácie selekcií a joinov: Nech R a S sú relačné výrazy. Nech F, G, H sú nekoneč predikáty také, že $\text{atrib}(F) \subseteq \text{atrib}(R)$, $\text{atrib}(G) \subseteq \text{atrib}(S)$ a $\text{atrib}(H) \subseteq \text{atrib}(R) \cup \text{atrib}(S)$. Potom
- $R \bowtie S \bowtie F \bowtie G \bowtie H \equiv ((R \bowtie F) \bowtie (S \bowtie G)) \bowtie H$
 - $\sigma_{F \wedge G \wedge H}(R \bowtie S) \equiv \sigma_H(\sigma_F R \bowtie \sigma_G S)$
 - $\sigma_{F \wedge G \wedge H}(R \times S) \equiv \sigma_H(\sigma_F R \times \sigma_G S)$

Zákony relačnej algebry – agregácia

11. Agregácia a join

- Ak $B = A \cap \text{atrib}(R)$ obsahuje nadkľúč v S a $x \in \text{atrib}(R)$.

$$\Gamma_{A,q \leftarrow \text{agg}(x)}(R \bowtie S) \equiv \Pi_A((\Gamma_{B,q \leftarrow \text{agg}(x)} R) \bowtie S)$$

12. Agregácia a projekcia

- Ak A obsahuje nakľúč R a $x \in A$, potom:

$$\Pi_A(\Gamma_{B,q \leftarrow \text{agg}(x)}(R)) \equiv \Gamma_{A,q \leftarrow \text{agg}(x)}(R)$$

Manipulácia s agregáčnymi výrazmi vyžaduje dodatočné informácie. Aspoň znalosť kľúčov resp. funkčných závislostí. Implementácie SQL preto väčšinou agregácie neoptimalizujú.

Dekompozícia selekčných predikátov

Pokiaľ je selekčný predikát v konjunktívnej forme zákony relačnej algebry ho umožňujú dekomponovať.

Čo s disjunkciou?

Príklady:

$$\sigma_{(x=a \vee y=b)} R(x,y) = \sigma_{x=a} R \cup \sigma_{y=b} R$$

$$\sigma_{(x=u \vee y=v)} R(x,y) \times S(u, v) = R \bowtie_{x=u} S \cup R \bowtie_{y=v} S$$

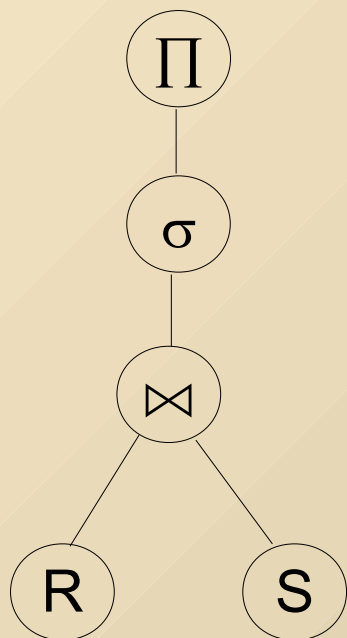
Relačná algebra ani SQL takéto pravidlá pri optimalizácii nepodporujú.

Pravidlá pre výpočet v relačnej algebre

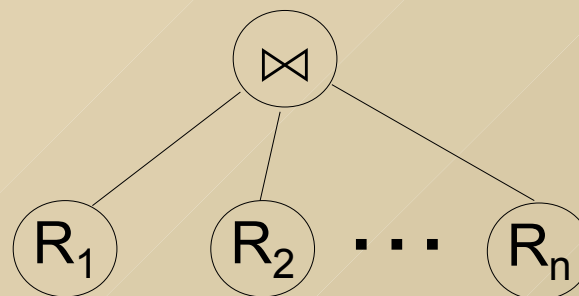
- A. Vykonaj selekciu čo najskôr. Obzvlášť selekcie na rovnosť. Tieto pravdepodobne najviac zmenšujú výsledok. (10)
- B. Vyhybaj sa kartézskym súčinom. Nahrad' ich joinami. Ak je aplikovaná selekcia na výsledok kartézského súčinu. Buď je to join, alebo selekcia môže byť posunutá na jednotlivé zložky kartézského súčinu.
- C. Kaskáda unárnych operácií (selekcií, projekcií) sa dá spojiť do jednej unárnej operácie. Prípadne môžeme túto unárnu operáciu spojiť s predošlou binárnou operáciou.
- D. Spoločné podvýrazy, ak ich veľkosť je rozumná, je výhodné si odložiť (prípadne aj do druhotnej pamäti). Ich zápis a čítanie môžu vyžadovať menej času ako opakované vyhodnotenie.

Strom výrazu

- Najjednoduchšie je znázorniť výrazy stromami.
- Vnútorne uzly reprezentujú operácie
- Listy relácie



V relačnej algebre sú unárne a binárne operácie. Faktor vetvenia 2. Ak je binárna operácia asociatívna a komutatívna je niekedy výhodné použiť jeden uzol reprezentujúci všetky možné zoskupenia.



Algoritmus optimalizácie

1. Použi pravidlo (4) na separáciu selekcií do kaskády:

$$\sigma_{F_1 \wedge \dots \wedge F_n}(E) = \sigma_{F_1}(\dots(\sigma_{F_n}(E)\dots))$$

2. Pre každú selekciu použi pravidlá (6, 8 a 10), aby si ju „prepasíroval“ čo najbližšie k listom.
3. Použi pravidlá (5, 7 a 12) na „prepasírovanie“ projekcií čo najbližšie k listom. Pozor na alternáciu selekcií a projekcií! Eliminuj zbytočné projekcie.
4. Spoj späť za sebou idúce projekcie a selekcie znovu do jednej projekcie a jednej selekcie.

Algoritmus optimalizácie, pokračovanie

5. Rozdel vnútorné uzly stromu do skupín, nasledovne:

- Každý vnútorný uzol reprezentujúci binárnu operáciu je v jednej skupine so svojimi predchodcami označenými unárnou operáciou (Π , σ).
- Každá skupina obsahuje aj postupnosť unárných potomkov, končiacich listom.
- Ak určujúci binárny uzol je kartézsky súčin (nie je predchádzaný vhodnou selekciou) jeho potomkov už do jeho skupiny nepridávame.

6. Výsledný plán môže počítat' skupiny v ľubovoľnom poradí, rešpektujúcim ich čiastočné usporiadanie v strome.

Príklad

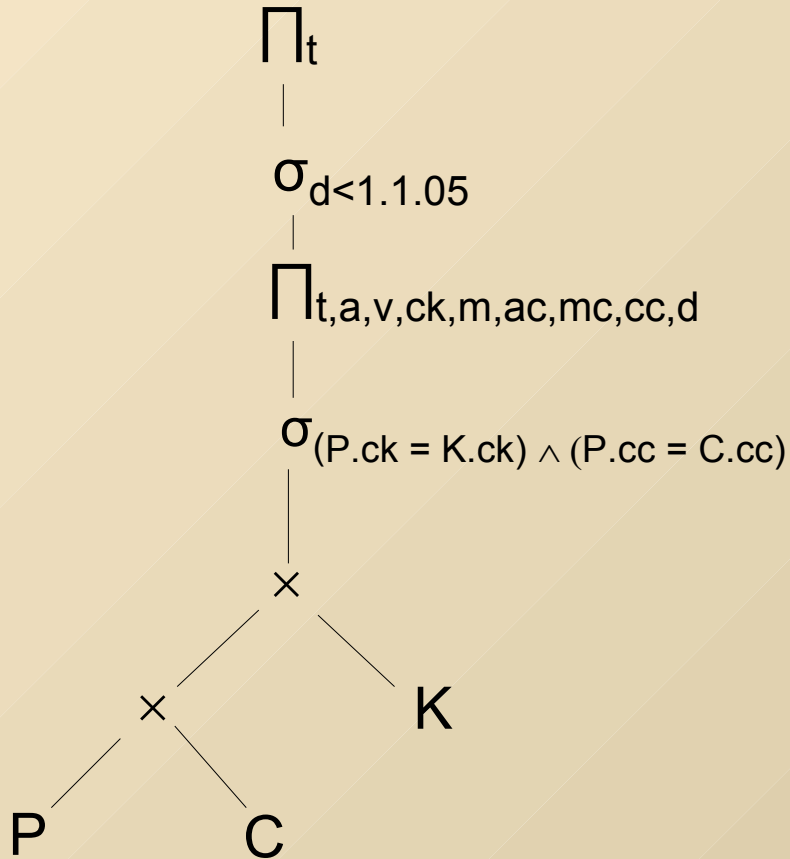
K(t, a, v, ck) – Kniha(titul, autor, vydavateľ, číslo_k)
V(v, av, mv) – Vydavatel(a, vydavateľ, adresa_v, mesto_v)
C(m, ac, mc, cc) – Čitateľ(meno, adresa_c, mesto_c, číslo_c)
P(cc, ck, d) – Požičal_si(číslo_c, číslo_k, dátum)

VL(t,a,v,ck,m,ac,mc,cc,d) \leftarrow P(cc,ck,d), C(m,ac,mc,cc), K(t,a,v,ck)
A(t) \leftarrow VL(t,a,v,ck,m,ac,mc,cc,d), d < 1.1.2005

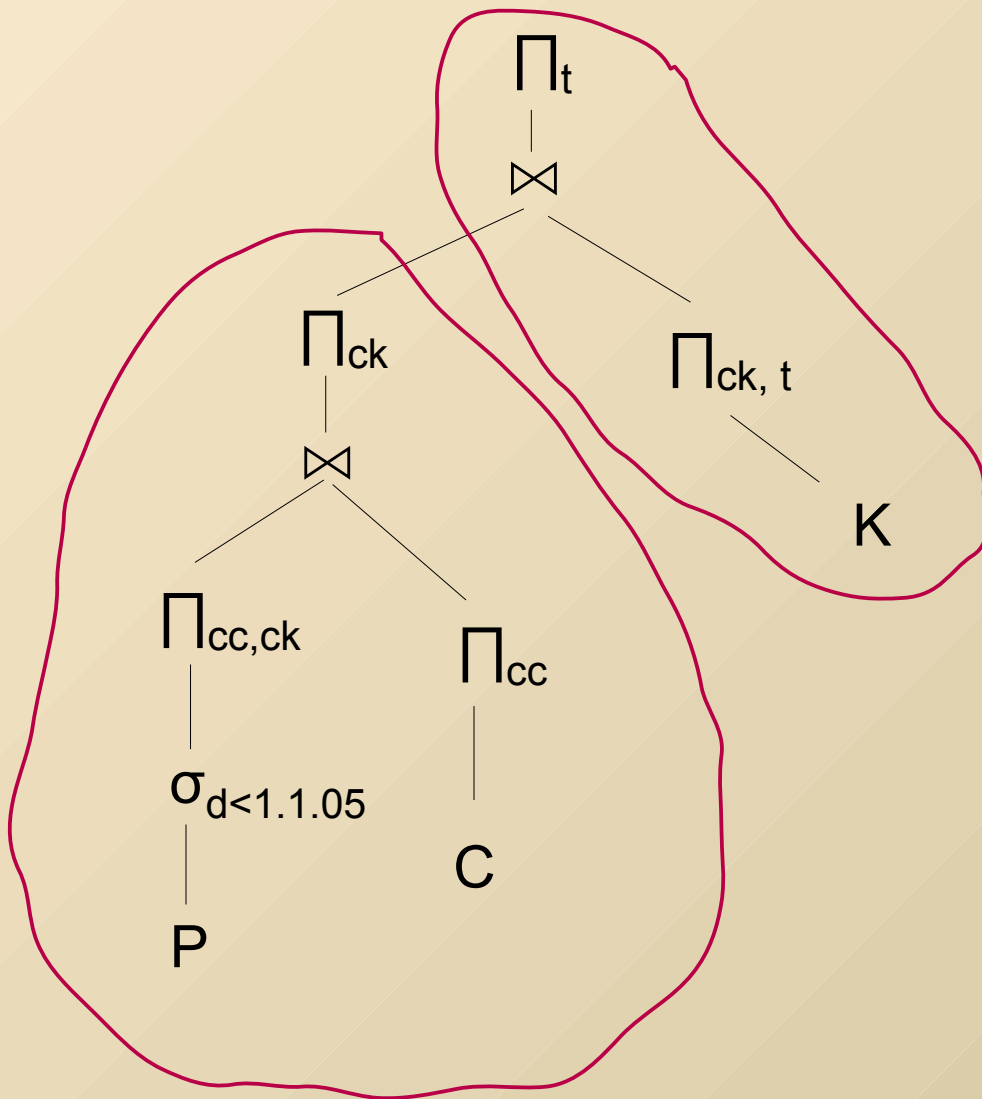
```
create view VL as select K.t,K.a,V.v,K.ck,C.m,C.ac,C.mc,C.cc,P.d  
                          from P, C, T  
                          where P.cc = C.cc and P.ck = K.ck;  
select t from VL where d < 1.1.2005;
```

Na prvý pohľad sa to zdá nelogické (dotaz sa dá ľahko formulovať aj priamo), ale to view zodpovedá výpožičnému lístku a „tetušky“ z knižnice vytvorí dotaz cez grafický interface.

Príklad – strom dotazu



Príklad – po optimalizácii



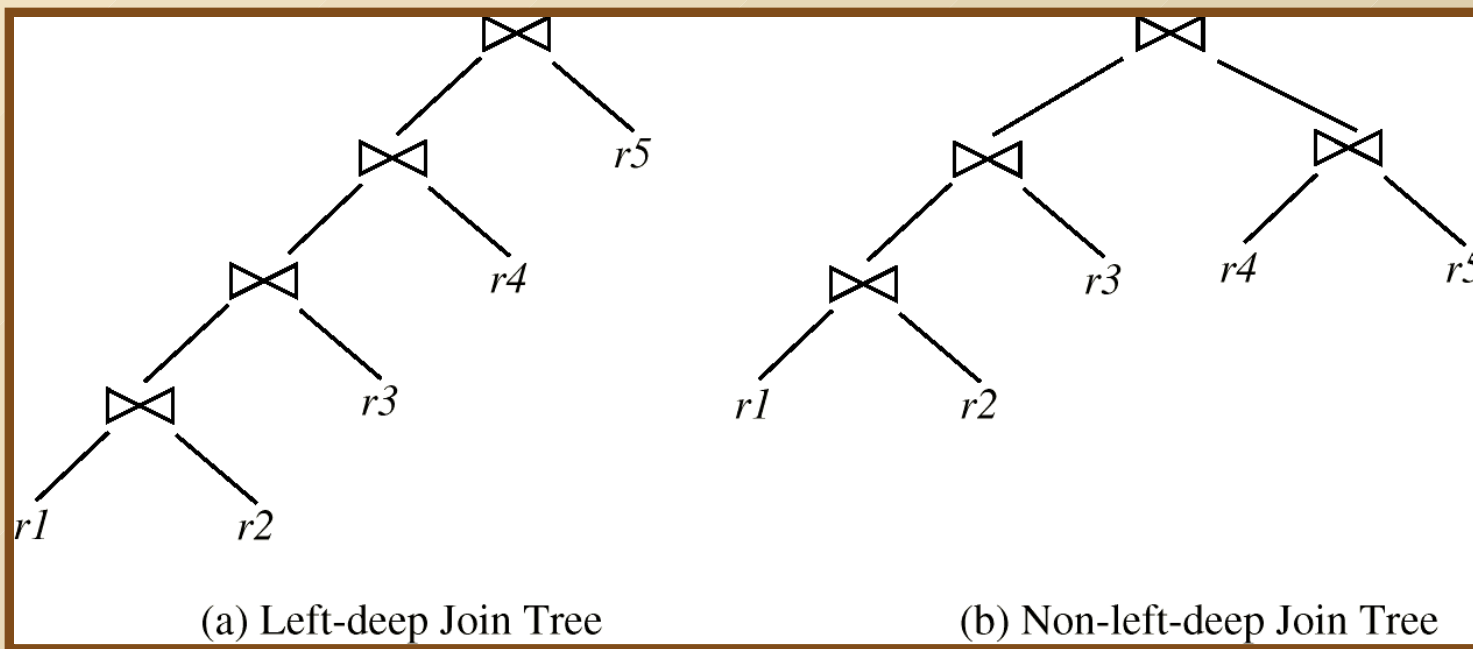
Na prvý pohľad sa zdá, že vnorená komponenta sa dá zjednodušiť na:

$\Pi_{ck}(\sigma_{d < 1.1.05}(P))$.

Nie je to celkom pravda. Uvedené dotazy sú ekvivalentné iba v prípade, že databáza neobsahuje výpožičné lístky na neexistujúcich čitateľov.

Poradie (štruktúra) joinov

V ľavolineárnych (left-deep) a pravolineárnych spájacích stromoch (join trees) je vždy jeden operand EDB relácia a druhý výsledok predošlých joinov.



Počet usporiadaní joinov

Chceme nájsť najlacnejšie usporiadanie joinov

$$r_1 \bowtie r_2 \bowtie \dots \bowtie r_n.$$

Existuje $(2(n-1))!/(n-1)!$ rôznych usporiadaní (je to počet binárnych stromov s n rôznymi listami). Pre $n = 7$, je ich 665 280. Pre $n = 10$, je to číslo rádove $10^{11} - 10^{12}$.

(Všade to tak uvádzajú, ale pretože join je komutatívny asi treba výsledok deliť 2^{n-1} . Celé sa to dá odhadnúť pomocou Stirlingovho vzorca na $2^{n-1}(n-1)!/(\pi(n-1))^{1/2}$.)

To sa zdá byť prakticky nepoužiteľné, aj keby sme nebrali do úvahy iné operácie.

Našťastie, nie sme nútený generovať všetky binárne stromy. Použijeme dynamické programovanie.

Vypočítame cenu najlacnejšieho usporiadania pre každú podmnožinu $\{r_1, r_2, \dots, r_n\}$ iba raz a zapamätáme si ju.

Zložitosť je takto „len“ $O(2^n)$.

Dynamické Programming

Najdenie najlepšieho plánu na spojenie n relácií:

Aby sme našli najlepší plán joinu množiny S , pozostávajúcej z n relácií, budeme uvažovať všetky plány tvaru: $S_1 \bowtie (S - S_1)$, kde S_1 je neprázdna podmnožina subset S .

Rekurzívne počítame ceny spájania podmnožín S , aby sme našli cenu každého plánu. Vyberieme najlepšíu (minimálnu) z $2^n - 1$ alternatív.

Aby sme sa vyhli zbytočnému počítaniu.

Zapamätáme si najlepší plán a minimálnu cenu pre každú už vypočítanú podmnožinu.

(Patricia G. Selinger, IBM project R)

Optimalizačný algoritmus – dynamické programovanie

```
procedure findbestplan(S);
{ if (bestplan[S].cost  $\neq \infty$ ) then return bestplan[S]
  else /* bestplan[S] has not been computed earlier, compute it now */
  { for each non-empty subset  $S_1$  of S such that  $S_1 \neq S$  do
    {  $P_1 := \text{findbestplan}(S_1)$ 
       $P_2 := \text{findbestplan}(S - S_1)$ 
      A := algorithm for joining results of  $P_1$  and  $P_2$ 
      cost =  $P_1.\text{cost} + P_2.\text{cost} + \text{cost of } A$ 
      if cost < bestplan[S].cost then
        { bestplan[S].cost = cost;
          bestplan[S].plan = “execute  $P_1.\text{plan}$ ; execute  $P_2.\text{plan}$ ;
            join results of  $P_1$  and  $P_2$  using A” }
        }
      return bestplan[S]
    }
  }
```

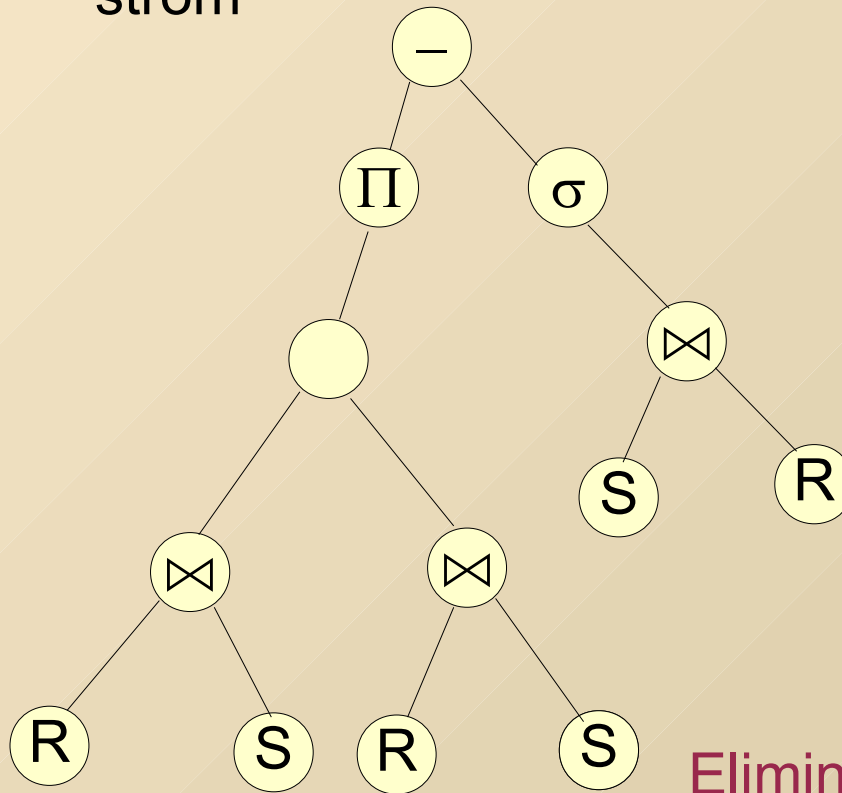
Kritika

- Mnohí považujú takúto optimalizáciu za pridrahú
- Navyše rozvetvené stromy vyžadujú pamäť pre medzivýsledky.
- Často sa preto používa len ľavo lineárny (pravo lineárny) výpočet.
- Poradie joinov sa vyberá „greedy“ metódou.

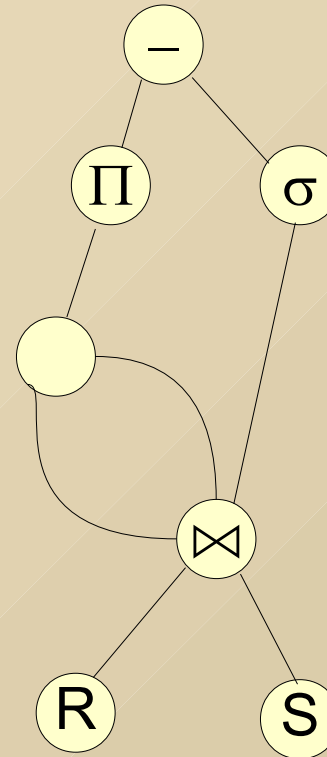
Výsledok takejto jednoduchkej optimalizácie môže byť ľubovoľne vzdialený od optima. V praxi je takýto jav dosť vzácny a často rozdiel nie je veľký. Možno sa vyplatí šetriť na optimalizácii.

Eliminácia spoločných podvýrazov

strom



dag



Eliminácia spoločných podvýrazov
spočíva v transformácii stromu na dag
(s minimálnym počtom uzlov).

Algoritmus eliminácie

Myšlienka algoritmu eliminácie je prekvapujúco jednoduchá a spočíva v opakovaní jediného cyklu.

repeat

find nodes with equal labels and the same successors;

/* be careful about the order of the successors */

make such nodes identical;

until no coincidence;

- Najprv sa stotožnia rovnako označené listy (majú rovnakých žiadných synov).
- Algoritmus môže eventuálne pokračovať, pretože už niektoré uzly majú tých istých synov.
- Ak uzly sú označené komutatívnou operáciou na poradí synov nezaleží. Inak sa synovia považujú za rôznych, aj keď sú v inom poradí.

Detail algoritmu – stotožnenie uzlov

- Abstraktná (vizuálna) predstava je, že uzol sa bude posúvať tak, aby splynul s uzlom, s ktorým sa má stotožniť. Hrany sa pri tom deformujú.
- Implementačná predstava:
 - Node { id:__; label: operation; leftson: rightson: linktonode }
 - Zo stotožňovaných uzlov vyberieme uzol s najmenším id.
 - Všetky smerníky z rodičov stotožňovaných uzlov presmerujeme na tento uzol.
 - Uzly, do ktorých nevchádza žiadna hrana odstránime. S odstránením uzla sa odstránia aj hrany z neho vychádzajúce.

Syntéza

- Riešili sme jednotlivé úlohy
 - Presun unárnych zmenšujúcich operácii k listom.
 - Poradie komutatívny a asociatívnych operácií.
Spojenia a kartézske súčiny. Pre zjednotenia je skoro jedno v akom poradí ich „nerobíme“.
 - Eliminácia spoločných podvýrazov.
- Optimalizáciu je vhodné robiť tak, že začneme so stromom, kde komutatívne a asociatívne operácie sú zlúčené do jedného uzla. Ako prvé pretlačíme selekcie a projekcie, čo najbližšie k listom. Potom zlúčime selekcie a kartézske súčiny do joinov. Určíme poradie joinov a nakoniec eliminujeme spoločné podvýrazy.

Prirodzený strom pre SQL

(bez vnorených poddotazov)

sort	order by
\cup	union
σ	having
Γ	agregácia
Π	select
σ	where
\times	from

V prípade vnorených poddotazov sa selekcie pre where a having nahrádzajú **joinom** resp. **antijoinom**, ktorého jedna vetva obsahuje pokračovanie dotazu a zvyšné vetvy vnorené poddotazy (okrem záverečného sortu). Proces vnárania môže rekurzívne pokračovať.

Ignorovali sme rozdiel (except), ale ten sa dá vždy nahradiť vnoreným poddotazom (not in / not exist). Mnohé implementácie SQL ho preto ani neobsahujú.

Redundancia relačnej algebry

- Relačná algebra je príliš „barokná“. Obsahuje priveľa operácií.
- Logickej spojke \wedge (and) zodpovedajú až tri operácie:
 - Selekcia σ , ak sa jedná o zabudovaný (matematický) predikát.
 - Join \bowtie , ak sa jedná o DB predikáty so spoločnými premennými.
 - Kartézsky súčin \times , ak sa jedná o DB predikáty bez spoločných premenných.
- Vlastne štyri.
 - Antijoin \ltimes , ak sa jedná o DB predikát a negovaný DB predikát.

Minimalizácia relačnej algebry

- Dilemou je: či je lepšie mať veľa elementárnych operácií, alebo málo všeobecných.
- Obe rozhodnutia majú svoje výhody a nevýhody.
- Pokus o málo všeobecných:
 - $\Pi, \bowtie, \Join, \Gamma, \cup$
 - Join používame tak, že v prípade disjunktných množín atribútov (premenných) operandov je to kartézsky súčin, v prípade rovnakých množín atribútov prienik, inak prirodzené spojenie.
 - Antijoin je len o málo všeobecnejší než rozdiel.
 $R \Join S = R - (R \bowtie S).$
 - Ostatné operácie zostávajú ako v relačnej algebre.

Iné modifikácie relačnej algebry

- Joiny
 - ľavý, pravý ľavopravý join
 - Θ join. Symbolicky $R \bowtie_{\Theta} S = (R \bowtie S) \bowtie \Theta$
- Relácie ako multimnožiny (duplikáty)
 - fuzzy relácie
 - pravdepodobnostné relácie
 - Databases with uncertainty and lineage
(Omar Benjelloun, Anish Das Sarma, Alon Halevy, Jennifer Widom)

Ohraničenia

Zovšeobecnené relácie vyžadujú isté ohraničenia. Tieto ohraničenia sa týkajú väzby premenných. V datalógu museli byť formuly bezpečné. V relačnej algebre a SQL sme zabudované (matematické) predikáty používali formou selekcie. Táto požaduje, aby všetky premenné, ktoré sa vyskytujú v selekčnom predikáte boli viazané. Ten istý požiadavok je kladený aj na rozdiel a agregáčnú funkciu. Z hľadiska optimalizácie sú tieto obmedzenia možno zbytočne silné a bránia niektorým optimalizáciám.

Príklad: $\sigma_{z=x+y}(R(x,y) \times S(z))$; $(R(x,y) \bowtie S(z)) \bowtie \text{add}(x,y,z)$

Poradie: $(R(x,y) \bowtie \text{add}(x,y,z)) \bowtie S(z)$ bolo zakázané. Je to však dobre realizovateľný a v prípade, že relácia R obsahuje niekoľko málo dvojíc a relácia S je veľká, je to optimálny spôsob vyhodnotenia uvedeného výrazu.

SQL: **with** sup(x,y,z) **as** **select** R.x, R.y, x+y **as** z **from** R
select sup.x, sup.y, sup.z **from** sup, S **where** S.z = sup.z ;

Predpoklady (výpočtový model)

Budeme predpokladať, že pre každý predikát v databáze sú definované minimálne povolené ozdoby. To je to isté ako minimálne podmnožiny atribútov (budeme ich nazývať pseudokľúče, input sets), pre ktoré v danej relácii existuje len konečný počet riadkov a databázový stroj ich vie efektívne vrátiť.

Príklady:

Pre EDB relácie je pseudokľúč \emptyset . (full scan)

Pre $\text{eq}(x,y) \equiv x=y$ sú to $\{x\}$, $\{y\}$.

Pre zmienený predikát $\text{add}(x,y,z) \equiv z=x+y$ to môžu byť všetky dvojice atribútov.

Predpokladáme, že v najhoršom je pseudokľúčom množina všetkých atribútov.

Uskutočniteľnosť operácií

- Unárne operácie
 - Π – projekcia vyžaduje aspoň jeden **pseudokľúč** operandu.
 - Γ – agregácia vyžaduje, aby grupovacie atribúty obsahovali **pseudokľúč** operandu.
- Binárne operácie
 - \cup – zjednotenie, bezpodmienečne aspoň jeden **pseudokľúč** pre každý operand.
 - \bowtie – join, **pseudokľúč** pre každý operand v čase spájania.
 - \ltimes – antijoin aspoň jeden **pseudokľúč** prvého operandu.
Druhý operand musí byť na základe spoločných atribútov aspoň rozpoznávač.

Modus operandi

Budeme predpokladať, komponentu výrazu vyhodnocujeme takým spôsobom, že si v nejakej postupnosti pomocných relácií zbierame čiastočné výsledky. Poradie operácií je len čiastočne určené, resp. môžeme aplikovať zákony relačnej algebry. Zaujíma nás, kedy môžeme, ktorú operáciu použiť. Určujú to premenné, ktoré už vieme – viazané premenné.

Vlastne pre každý predikát máme určené minimálne dovolené ozdoby. Je to zaujímavé pre zabudované a negované predikáty. Pretože, „kľúč“ je tu skôr odraz úmyslu programátora než reálny kľúč relácie. Tie operácie, ktoré ho požadujú dovoľujeme volať len pre predikáty s ozdobou $b...b$.

Znázornenie hypergrafom

- Grafy len operácie a operandy. Nedokážeme sledovať premenné a väzby.

Hypergraf

- Uzly zodpovedajú premenným
- Hrany zodpovedajú predikátom
 - nenegovaným DB predikátom
 - zabudovaným (matematickým) predikátom
 - predikátu rovnosti
 - agregáčným predikátom
 - negovaným predikátom

Optimalizácie pomocou redukcie hypergrafu

Ján Šturc
Jar, 2013

Reprezentácia dataológového pravidla hypergrafom

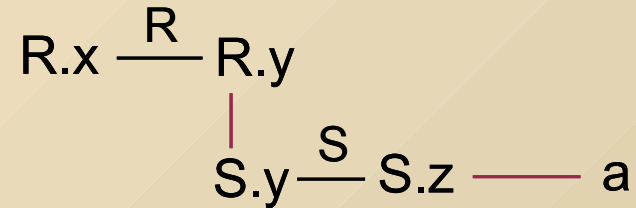
- Uvažujeme pravidlo $p \leftarrow q_1(x_{1,1} \dots x_{1,k_n}), \dots, q_n(x_{n,1} \dots x_{n,k_n})$.
- Niektoré premenné môžu eventuálne byť rovnaké, alebo byť konštanty.
- Premenným a konštantám priradíme uzly hypergafu.
- Tie premenné a konštanty, ktoré sa vyskytujú v jednom predikáte (nenegovanom, alebo negovanom) spojíme hranou a pomenovanou príslušným predikátom (q , a $(\neg q)$) sú rôzne hrany.
- Hrany zodpovedajú DB predikátom konečným, alebo selekčným (zabudovaným, matematickým) potenciálne nekonečným a agregáčnym predikátom. Zo selekčných predikátov špeciálne vyčleňujeme **rovnosť**.
- **Spolu 7 typov hrán.**

Reprezentácia select príkazu hypergrafom

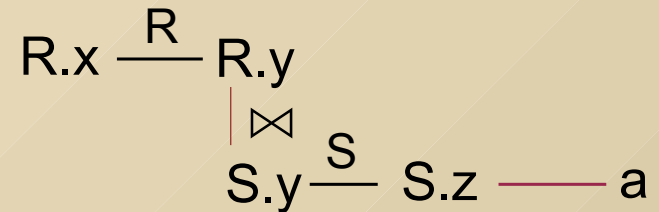
- V SQL je situácia jednoduchá. Všetky premenné sa vyskytujú v DB hranách zodpovedajúcich EDB predikátom a majú rôzne mená.
- Rovnosti premenných sú explicitne určené vo where (alebo join) klauzule.
- Selekčné podmienky zodpovedajú jednotlivým konjunktom (ktoré nie sú rovnosti premenných) where klauzy.
- Jednoduchý selekt neobsahuje iné hrany.
- Vnoreným selektom zodpovedá nový jednoduchý selekt. Výsledná relácia sa „joinuje“ (in, exists), alebo „antijoinuje“ (not in, not exist).
- Premenné vnorených selektov môžu byť prepojené s premennými hlavného dotazu, alebo iných vnorených dotazov selekčnými hranami (aj rovnosti).

Príklad.

select R.x **from** R, S
where R.y = S.y and S.z = 'a'



select R.x **from** R
where R.y **in** (**select** S.y **from** S
 where S.z = 'a')



select R.x **from** R
where exists (**select** * **from** S
 where R.y = S.y
 and S.z = 'a')



$\emptyset \equiv \text{false}$

$\{\emptyset\} \equiv \text{true}$

Nie celkom jasné prečo sa v SQL manuáloch tvrdí, že join je rýchlejší (efektívnejší) ako vnorený dotaz ? Asi implementácie SQL prekladajú vnorený dotaz priamo na „nested loop“.

Transformácia na štandardný hypergraf.

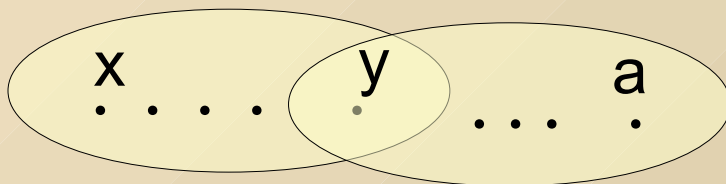
- V štandardnom hypergrafe sa nevyskytujú hrany rovnosti.
- Ich odstránenie je jednoduché. Ak sa rovnajú dve premenné, vyberieme jedného reprezentanta a obe transformujeme do tohto reprezentanta.
- Ak sa premenná rovná konštante, nahradíme ju konštantou. Na hranu aplikujeme príslušnú selekciu.
- Pri náhrade premenných urobíme všetky hrany s nimi incidentné, incidentné s jedinou inštanciou ich reprezentanta.
- Ak sa v zápise vyskytuje join (kartézsky súčin) nahradíme ho spojením podľa príslušných premenných
- Ak sa jedná o zložitejšie vnorené dotazy, vyžaduje to často pomocnú hranu pre medzivýsledky.

Príklad – pokračovanie

Všetky tri dotazy na slide 4 sa transformujú na štandardný hypergraf:

$$x \text{---} \underline{R} \text{---} y \text{---} \underline{S} \text{---} a$$

Ak v reláciach R a S existujú aj iné atribúty:



$$(\sigma_{z=a} S) \bowtie R \text{ alebo } R (\bowtie \circ \sigma_{z=a}) S$$

Optimalizácia jednoduchých dotazov (SPJ)

Budeme sa zaoberať dotazmi tvaru:

$$\Pi (\sigma_{F_1 \wedge \dots \wedge F_m} (R_1 \times \dots \times R_n))$$

Po našich transformáciach je to join:

$$\Pi (R_1 \bowtie \dots \bowtie R_n \bowtie F_1 \bowtie \dots \bowtie F_k)$$

Niektoré podmienky rovnosti pohltili joiny. Hypergraf obsahuje len dva typy hrán. Hrany zodpovedajúce pozitívnym EDB predikátom a hrany zodpovedajúce pozitívnym selekčným predikátom.

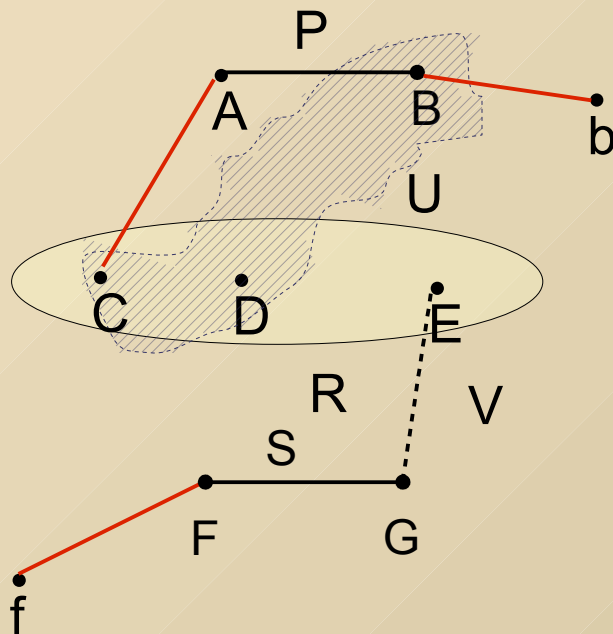
Nakoniec k takémuto dotazu pridáme záverečnú projekciu (select – from – where).

Príklad

Schéma: P(A,B), R(C, D, E), S(F, G)

select P.A, R.D, R.E, S.G **from** P, R, S

where ((P.A = R.C) **and** (P.B < R.C \vee P.B < R.D)
and (S.G < R.E) **and** (S.F="f") **and** (P.B="b"));

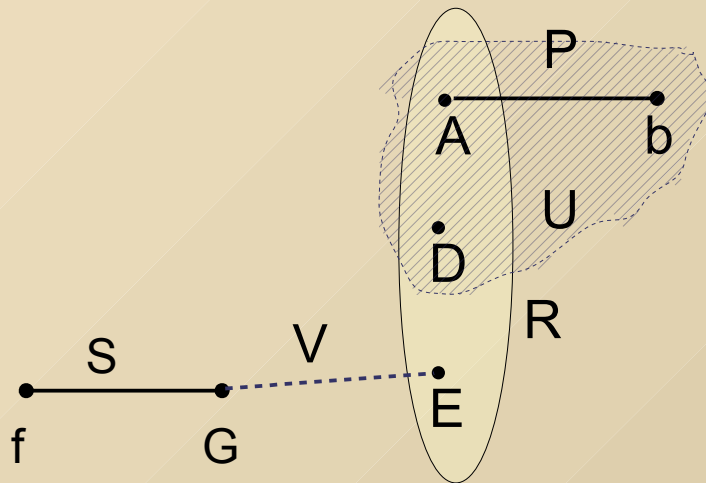


Príklad (pokračovanie)

$P(A,B), R(C,D,E), S(F,G)$

$$W \equiv (A = C) \wedge \underbrace{(B < C \vee B < D)}_U \wedge \underbrace{(G < E)}_V \wedge (F=f) \wedge (B=b)$$

Štandardný hypergraf:



Hrany:

$$P \equiv \sigma_{B=b} P$$

$$R \equiv \Pi_{C \leftarrow A, D, E} R$$

$$S \equiv \sigma_{F=f} S$$

U a V sú selekčné hrany

$$U \equiv A > b \vee D > b$$

Wong Youssefiho algoritmus

QUEL redukcia hypergrafu

1. Ak hypergraf G je zjednotenie viacerých „nesúvislých“ komponent $\mathcal{H}_1 \dots \mathcal{H}_k$, potom
 - $\text{prog}(G)$ je $\text{prog}(\mathcal{H}_1); \dots; \text{prog}(\mathcal{H}_k)$;
 - $\text{res}(G) := \text{res}(\mathcal{H}_1) \times \dots \times \text{res}(\mathcal{H}_k)$
2. Ak v hypergrafe G eliminujeme selekčnú hranu U a výsledný hypergraf je \mathcal{H} . Potom
 - $\text{prog}(G)$ je $\text{prog}(\mathcal{H})$;
 - $\text{res}(G) := \sigma_U(\text{res}(\mathcal{H})) \quad (\text{res}(\mathcal{H}) \bowtie U)$
3. Ak po odstránení relačnej hrany R sa hypergraf G rozpadne na k súvislých komponent ($k=1$ je možné.) $\mathcal{H}_1 \dots \mathcal{H}_k$, potom
 - $\text{prog}(G) =$ **for** each EDB edge S that intersect R **do**
 $\{S := S \times R\}; \quad \text{prog}(\mathcal{H}_1); \dots; \text{prog}(\mathcal{H}_k)$;
 - $\text{res}(G) := R \bowtie \text{res}(\mathcal{H}_1) \bowtie \dots \bowtie \text{res}(\mathcal{H}_k)$

Malé (redukované) relácie

poradie odstraňovania hrán

1. Reláciu R považujeme za malú, ak jej hrana obsahuje konštantu (selekcia tvaru $x = a$).
 2. Reláciu R považujeme za malú potom, čo jej n -tice boli redukované **semijoinom**.
- Vieme si predstaviť aj iné dôvody pre považovanie relácie za malú, tie sa ale v tejto optimalizácii neuplatnia.

Projekcia významné uzly

- Intuitívne významné (zaujímavé) premenné (uzly) sú tie, ktoré sa vyskytujú vo výsledku a tie, ktoré potrebujeme pri spájaní a testovaní podmienok.
1. Ak G je počiatočný hypergraf dotazu, potom uzol N je významný práve vtedy, keď sa vyskytuje vo výsledku.
 2. Ak hypergraf \mathcal{H} vznikol z G elimináciou hrany E . Potom uzol $N \in \mathcal{H}$ je významný práve vtedy, ak bol významný v G , alebo $N \in E$.
 3. Ak G je zjednotenie nesúvislých komponent $\mathcal{H}_1 \dots \mathcal{H}_k$. Potom uzol N je významný v G práve vtedy, ak je významný v niektorej z komponent \mathcal{H}_i .

Podmienky a preferencie pri eliminácii hrán

- Relačnú hranu môžeme odstrániť, ak pretína (má neprázdny prienik) s aspoň jednou inou relačnou hranou a nepretína žiadnu selekčnú hranu.
- Preferencie pri odstraňovaní relačných hrán:
 - (a) Ak existuje kandidát „malá relácia“, vyradíme všetkých kandidátov, „ktorí nie sú malí“.
 - (b) Ak niektorá zo zvyšných hrán rozbíja hypegraf na viac komponent súvislosti, vyradíme všetkých kandidátov, ktorí ho nerozbíjajú. (Preferencia multiway join.)
 - (c) Zo zvyšných kandidátov si vyberieme ľubovoľne.
- Ak nemôžeme odstrániť relačnú hranu musíme odstrániť selekčnú hranu. Vyberieme takú, aby sme po jej odstránení mohli použiť (a), ak taká neexistuje, tak aspoň (b). Ak ani taká neexistuje, uspokojíme sa s ľubovoľnou selekčnou hranou.

Multiway join – prečo rozbíjači ?

Treba vypočítať $R \bowtie S_1 \bowtie \dots \bowtie S_n$, predkladáme, že:

- ak $i \neq j$, potom S_i a S_j nemajú spoločné atribúty a
- pre všetky i má S_i nejaký spoločný atribút s R .

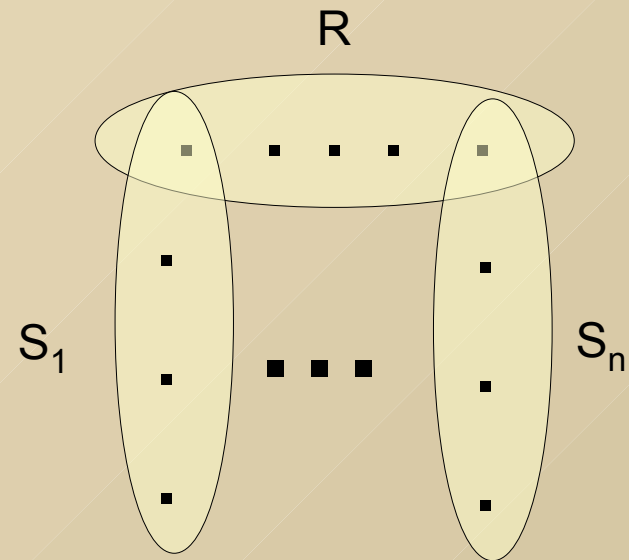
for each tuple $\mu \in R$ **do**

```
{ for  $i := 1$  to  $n$  do  $T_i := S_i \bowtie \{\mu\};$   
  output  $\{\mu\} \bowtie T_1 \bowtie \dots \bowtie T_n$  }
```

Jednoducho povedané. Pre n -ticu μ najprv určíme: s ktorými n -ticami sa spája.

Ak relácie majú vhodnú organizáciu (index), to môže byť rýchla operácia.

Ak sa s nejakou reláciou nespája, môžeme rovno prejsť na nasledujúcu n -ticu.



Wong Youssefiho algoritmus (QUEL)

Vstup: výraz tvaru $\Pi_D \sigma_F(R_1 \times \dots \times R_n)$

Výstup: program (strom) na výpočet výrazu

Metóda: rekurzívna procedúra *EVAL* riadená úpravou a dekompozíciou hypergrafu. Zostrojíme štandardný hypergraf výrazu G (Konštrukcia transformuje vstupné relácie na podvýrazy zodpovedajúce hranám.) a zavoláme *EVAL*(G, D).

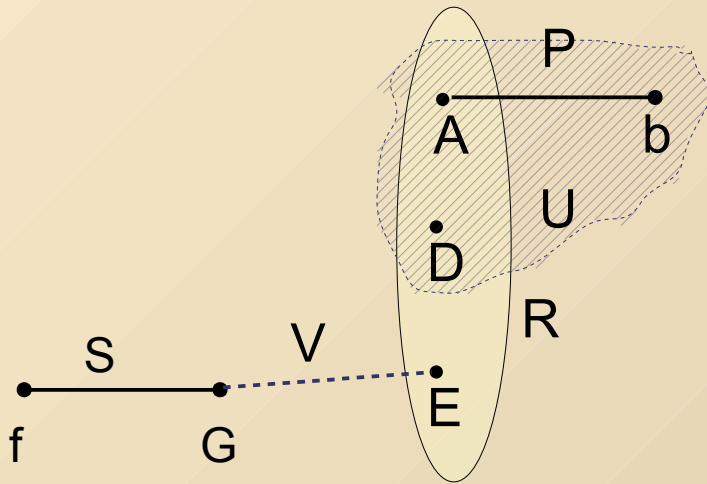
Procedúra *EVAL* si vyberá pre dekompozíciu hrany podľa pravidiel preferencie uvedených na predošlých slidoch.

Na výslednom strome ešte zlúčime binárne operácie s predchádzajúcimi unárnymi operáciami.

Zhustená schéma procedúry *EVAL*

- Prípád disjunktných komponent
 for $i:=1$ **to** k **do** { $E_i := \text{nodes}(\mathcal{H}_i)$; $EVAL(\mathcal{H}_i, E_i)$ }
 emit " $\text{res}(G) := \text{res}(\mathcal{H}_1) \times \dots \times \text{res}(\mathcal{H}_k)$ "
- Odstránenie relačnej hrany R
 for each EDB edge S that intersect R **do** emit " $S := S \bowtie R$ ";
 – **for** $i:=1$ **to** k **do** { $E_i := (D \cup R) \cap \text{nodes}(\mathcal{H}_i)$; $EVAL(\mathcal{H}_i, E_i)$ }
 emit " $\text{res}(G) := \Pi_D(R \bowtie \text{res}(\mathcal{H}_1) \bowtie \dots \bowtie \text{res}(\mathcal{H}_k))$ "
- Odstránenie selekčnej hrany F
 $E := (D \cup F) \cap \text{nodes}(\mathcal{H})$;
 $EVAL(\mathcal{H}, E)$;
 emit " $\text{res}(G) := \Pi_D(\sigma_F(\text{res}(\mathcal{H})))$ "

Príklad (dokončenie)



Hrany:

$$P' \equiv \sigma_{B=b} P$$

$$R \equiv \prod_{C \leftarrow A, D, E} R$$

$$S' \equiv \sigma_{F=f} S$$

U a V sú selekčné hrany

$$U \equiv A > b \vee D > b$$

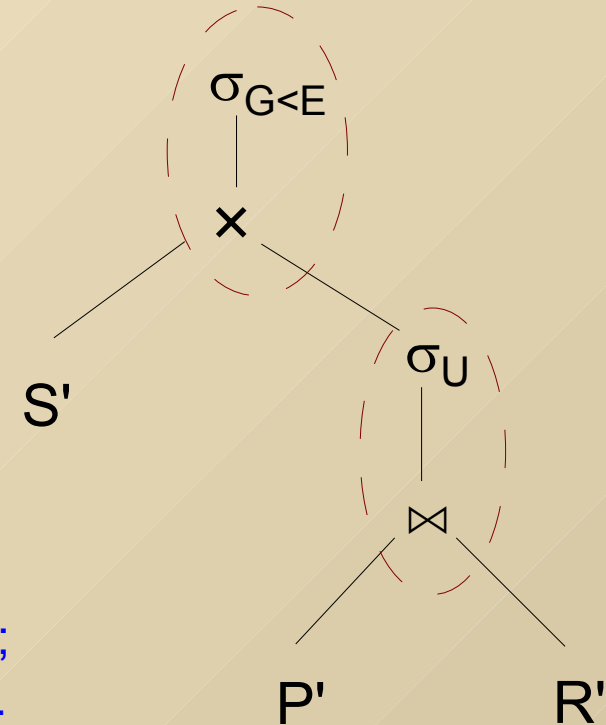
Program:

$$P' := \sigma_{B=b} P;$$

$$S' := \sigma_{F=f} S;$$

$$R' := R \rtimes P';$$

Result: $S'(x \circ \sigma_{G \leq E})(P'(\bowtie \circ \sigma_U)R')$



Veľkosť joinu

- Uvažujme join $A_1A_2 \bowtie A_2A_3 \bowtie \dots \bowtie A_{n-1}A_n$.
- Nech každý atribút A_i má obor definície $\{1, 2, 3, 4\}$.
- Nech každá relácia pozostáva z ôsmych dvojíc takých, že každá relácia obsahuje všetky dvojice s nepárnym súčtom (tvaru $\langle o, e \rangle$ alebo $\langle e, o \rangle$).
- Výsledok joinu pozostáva zo všetkých n -tíc $A_1A_2 \dots A_n$ takých, že sa v nich striedajú nepárne a párne čísla. Spolu 2^{n+1} n -tíc.
- Hoci celková veľkosť $n - 1$ relácií na vstupe je $8 \times (n - 1)$.

Ak chceme rozumne oceniť zložitosť výpočtu joinu, musíme ju oceňovať v termínoch veľkosti vstupu aj výstupu.

Reduktory – úplný reduktor

- Je možné, že výpčet $R_1 \bowtie R_2 \bowtie \dots \bowtie R_n$ bude prebiehať takým spôsobom, že najprv budú medzivýsledky narastať a nakoniec sa zredukujú. Vstup malý, výsledok malý, ale veľká priestorová a výpočtová náročnosť.
- Reduktorom nazývame program pozostávajúci s postupnosti príkazov tvaru $R := R \bowtie S$.
- Reduktor slúži na redukciu počtu n -tíc, ktoré vchádzajú do výpočtu.
- Úplný reduktor je taký reduktor, ktorý zabezpečuje, že do výpočtu joinu nevôjde žiadna zbytočná n -tica.

Db verzia GYO redukcie – trhanie uší (ear removing)

Definícia: Hrana E hypergrafu \mathcal{H} sa nazýva ucho, ak je jediná izolovaná hrana hypergrafu, alebo v hypergrafe \mathcal{H} existuje, taká hrana F , svedok uchovitosti pre hranu E , že vrcholy $E - F$ nie sú incidentné so žiadnou inou hranou hypergrafu \mathcal{H} .

Postupné trhanie uší, pokiaľ nezostane prázdny hypergraf je pre redukovateľné hypergrafy ekvivalentné GYO redukcii. Svedok uchovitosti F hrany E je tá hrana, ktorá pokryje (bude nadmnožinou) hranu E , ak z hrany E odstránime vrcholy, ktoré sú incidentné len s hranou E .

Bezstrátové spojenia

- $\Pi_{R_i}(R_1 \bowtie \dots \bowtie R_n) = R_i$ (1) global consistency
- Pre každé i, j platí $\Pi_{R_i}(R_i \bowtie R_j) = R_i$ (2) local consistency

Veta: $(1) \Rightarrow (2)$.

Ak hypergraf $R_1 \bowtie \dots \bowtie R_n$ je GYO redukovateľný,
potom $(2) \Rightarrow (1)$.

Dôkaz:

Prvé tvrdenie dostaneme podľa pravidla 9 (slide 12)
z predošlej prednášky dosadením výrazu $R_1 \bowtie \dots \bowtie R_n$
za R a S a položením $A=B=R_i$ a $C=R_j$.

Dôkaz druhého tvrdenia

Indukciou vzhľadom na počet relácií.

1. Pre $n=1$ a $n=2$ nie je čo dokazovať.
2. Predpokladajme, že program platí, pre $n-1$ relácií dokážeme, že platí pre n .

Hypergraf je acyklický, teda v ňom existuje ucho E a svedok uchovitosti F . Po odstránení ucha E zvyšný hypergraf H obsahuje $n-1$ relácií. Medzi nimi aj reláciu zodpovedajúcu hrane F . Tieto relácie sa spájajú bezstrátovo.

Každý riadok každej hrany (relácie) hypergrafu H sa podiela na výsledku.

Hrany E a F sa spájajú bezstrátovo podľa predpokladu.

Teda každý riadok E sa spája s nejakým riadkom relácie pre H . Atribúty, ktoré nie sú v F to nemôžu ovplyvniť, lebo sa vyskytujú iba v E .

Výpočet úplného reduktoru

- Znázorníme join hypergrafom
- Aplikujeme nasledujúcu rekurzívnu procedúru:

```
procedure reduce(G);  
{ find an ear E with witness F;  
  emit "F := F  $\bowtie$  E;";  
  reduce(G-{E});  
  emit "E := E  $\bowtie$  F;"  
}
```

Príklad - jednoduchý

A	B	B	C	C	D
1	2	1	2	1	2
2	4	2	4	2	4
3	6	3	6	3	6
4	8	4	8	4	8

Reduktor:

$AB := AB \bowtie BC;$

$BC := BC \bowtie CD;$

$CD := CD \bowtie BC;$

Úplný reduktor:

$BC := BC \bowtie AB;$

$CD := CD \bowtie BC;$

$BC := BC \bowtie CD;$

$AB := AB \bowtie BC;$

Pre acyklické joiny úplný reduktor obsahuje $2 \times (k-1)$ príkazov priradenia a jeho generovanie je nezávislé od obsahu relácií.

Počas vyhodnotenia úplného reduktoru, žiadna relácia nerastie (často sa zmenší). Zložitosť vyhodnotenia úplného reduktoru je polynomiálna.

Príklady – cyklické

A	B	B	C	A	C
0	0	0	0	0	1
1	1	1	1	1	0

Každé dve relácie sa spájajú bezstrátovo.

Spojenie všetkých troch je prázdne. Žiaden semijoin nič neredukuje.

A	B	B	C	A	C
1	1	1	1	2	1
2	2	2	2	3	2
...
n	n	n	n	n+1	n

1

Spojenie všetkých troch relácií je prázdne. Po i semijoinoch sa môžu vyredukovať iba dvojice obsahujúce čísla $k \leq i$ alebo $k > n-i$. „Úplný reduktor“ obsahuje $n/2$ príkazov priradení. Jeho veľkosť závisí od obsahu relácií.

Redukcia konkrétnej hrany

Veta: Redukovateľný hypergraf s viac ako jednou hranou má aspoň dve uši.

Dôkaz: Indukciou vzhľadom na počet hyperhrán.

Pre $n = 2$ to platí. Trivialne. $N = 3$ preskúmanie všetkých možností.

Predpokladajme, že pre hypergrafy s menej ako $n \geq 3$ hyperhranami to platí. Dokážeme, že to platí aj pre n . Nepriamo.

1. Nemá žiadne ucho spor s redukovateľnosťou.

2. Má jediné ucho E so svedkom uchovitosti F . Potom F je aj svedkom uchovitosti všetkým hranám, ktorým bolo svedkom uchovitosti E . Po odstránení E sa novým uchom, môže stať jedine F . To je spor s indukčným predpokladom.

Dôsledok: Môžeme si vybrať, ktorú hranu chceme odstrániť ako poslednú.

Yannakakisov algoritmus

Vstup: Relačný výraz $\Pi_X(R_1 \bowtie \dots \bowtie R_k)$ s acyklickým hypergrafom G a relácie R_1, \dots, R_k .

Výstup: Relácia hodnota výrazu $\Pi_X(R_1 \bowtie \dots \bowtie R_k)$

Metóda: pozostáva zo štyroch krokov

1. Zbav sa zbytočných atribútov a redukuj relácie pomocou úplného reduktoru.
2. Zostroj dekompozičný strom P hypergrafu G .
3. Traverzuj strom P v nejakom poradí od spodu nahor.
Ak relácia R je otec relácie S , urob $R := \Pi_{R \cup (X \cap S)}(S \bowtie R)$.
4. Vypočítaj záverečnú projekciu Π_X koreňa.

Spojenie prvých dvoch krokov – I. fáza

```
procedure reduce(G);  
{ find an ear E with witness F;  
  F :=  $\Pi_{Z \cap F}(F \bowtie E)$ ; /* Z sú zaujímavé atribúty */  
  emit "hrana(F,E)"; /* F je otec E*/  
  reduce(G-{E});  
  E :=  $\Pi_{Z \cap E}(E \bowtie F)$ ;  
}
```

Z sú zaujímavé atribúty, je to množina atribútov výsledku X zjednotená s množinou Y tých atribútov, podľa ktorých sa relácie spájajú.

Výsledkom tohto algoritmu je dekompozičný strom a redukované (bezstrátovo sa spájajúce) relácie.

Post-order traverzovanie – II. fáza

```
procedure traverse(R, Z);  
{ for i:=1 to k do { Y:= R $\cup$ (Z $\cap$ Si); traverse(Si, Y) }  
  /* S1, ..., Sk sú synovia uzlu R */  
  R :=  $\Pi_{R\cup Z}(R \bowtie S_1 \bowtie \dots \bowtie S_k)$  /* Z sú zaujímavé atribúty */  
}
```

Hlavný program je:

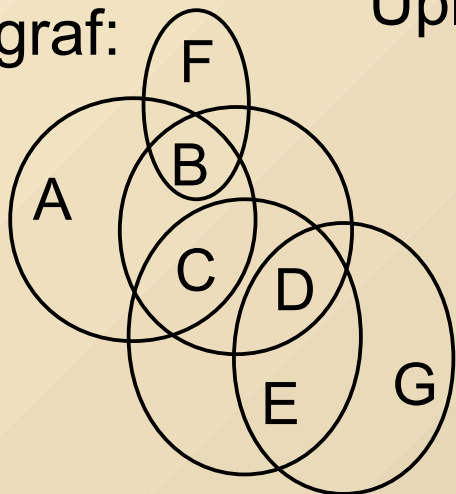
```
Traverse(root, root $\cup$ X);  
 $\Pi_X$ root;
```

Výpočet joinu Yannakakisovým je polynomiálny vzhľadom na veľkosť vstupu a výstupu.

Príklad

$$\Pi_{A,G}(ABC \bowtie BF \bowtie BCD \bowtie CDE \bowtie DEG)$$

hypergraf:



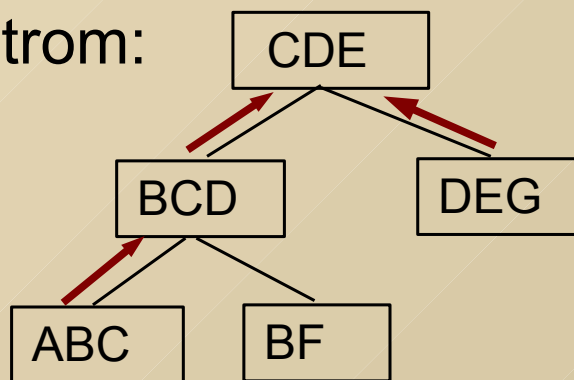
Úplný reduktor:

```
BCD:= BCD ⋈ ABC;
BCD:= BCD ⋈ BF;
CDE:= CDE ⋈ BCD;
CDE:= CDE ⋈ DEG;
DEG:= DEG ⋈ CDE;
BCD:= BCD ⋈ CDE;
BF := BF ⋈ BCD;
ABC:= ABC ⋈ BCD;
```

Redukcia: Ucho Svedok

ABC	BCD
BF	BCD
BCD	CDE
DEG	CDE

Strom:

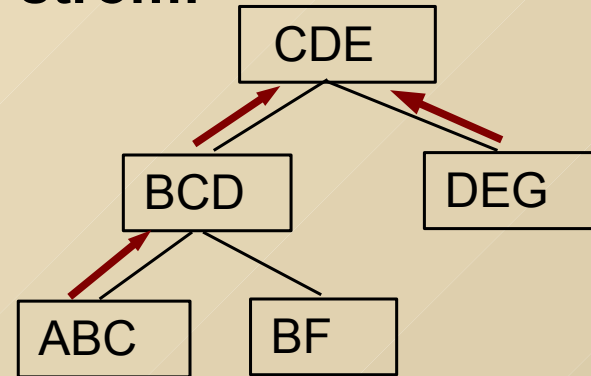


Príklad – pokračovanie

Naplnenie relácií po redukcii (priklad):

<u>ABC</u>	<u>BF</u>	<u>BCD</u>	<u>CDE</u>	<u>DEG</u>
$a_1b_1c_1$	b_1f_1	$b_1c_1d_1$	$c_1d_1e_1$	$d_1e_1g_1$
$a_2b_1c_1$	b_1f_2	$b_1c_1d_2$	$c_1d_2e_1$	$d_1e_1g_2$
			$d_2e_1g_1$	

strom:



Výpočet:

<u>ABCD</u>	<u>ACDE</u>	<u>ACDEG</u>	<u>AG</u>
$a_1b_1c_1d_1$	$a_1c_1d_1e_1$	$a_1c_1d_1e_1g_1$	a_1g_1
$a_1b_1c_1d_2$	$a_1c_1d_2e_1$	$a_1c_1d_1e_1g_2$	a_1g_2
$a_2b_1c_1d_1$	$a_2c_1d_1e_1$	$a_1c_1d_2e_1g_1$	a_2g_1
$a_2b_1c_1d_2$	$a_2c_1d_2e_1$	$a_2c_1d_1e_1g_1$	a_2g_2
		$a_2c_1d_1e_1g_2$	
		$a_2c_1d_2e_1g_1$	

Výsledok:

Zovšeobecnenia

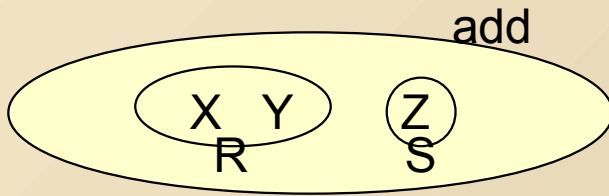
- Selekcie (zabudované predikáty)
 - doteraz pseudokľúč len všetky atribúty
 - využiť aj iné pseudokľúče
 - „zrovnoprávniť“ selekčné a relačné hrany
- Rozdiely
 - chýba niečo ako bezstrátovosť
 - skôr fungujúce heuristiky ako ucelená teória
 - zdá sa mi antijoin nádejnejší ako rozdiel
- Agregácie
 - mohlo by to byť zaujímavé (dátové kocky)

Plusy a úskalia prístupu selekcia je join.

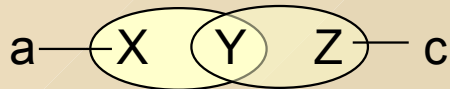
1

Dobré príklady:

$$\sigma_{Z=X+Y}(R(X,Y) \times S(Z)) = R(X,Y) \bowtie S(Z) \bowtie \text{add}(X,Y, Z)$$



$$\sigma_{X=a \wedge Z=c}(R(X,Y) \bowtie S(Y, Z)) = \text{eq}(X,a) \bowtie \text{eq}(Z,c) \bowtie R(X,Y) \bowtie S(Y, Z)$$

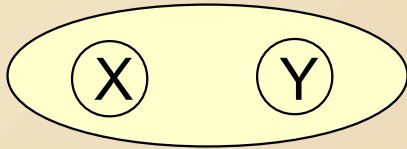


Plusy a úskalia prístupu selekcia je join.

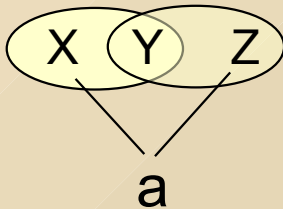
2

Zlé príklady:

$$\sigma_{x < y}(R(X) \times S(Y)) = R(X) \bowtie S(Y) \bowtie \text{less}(X, Y)$$



$$\sigma_{X=a \wedge Z=a}(R(X, Y) \bowtie S(Y, Z)) = \text{eq}(X, a) \bowtie \text{eq}(Z, a) \bowtie R(X, Y) \bowtie S(Y, Z)$$



Práca s nekonečnými reláciami

1. Všetkým inštanciam konštant priradiť rôzne uzly (aj keď niektoré nadobúdajú rovnaké hodnoty).
2. Počas redukcie s každou hranou eviduj dosiahnuté atribúty, sú to atribúty ktoré zodpovedajú viazným atribútom vo výpočte.
3. Na počiatku sú dosiahnuté atribúty hyperhrán:
 - a) Všetky atribúty hrany pre konečné (EDB) relácie.
 - b) Žiadne (prázdna množina) pre nekonečné relácie.
4. Vždy, keď sa hrana stane svedkom uchovitosti, pridaj k množine jej dosiahnutých atribútov, (zaujímavé) atribúty odtrhnutého ucha.
5. Ak je nekonečná relácia svedkom uchovitosti alebo uchom. Ucho možno odtrhnúť iba, ak zjednotenie dosiahnutých atribútov ucha a svedka uchovitosti pokrýva nejakú vstupnú množinu (dovolenú ozdobu) tejto nekonečnej relácie.

Neredukovateľné hypergrafy

- Wong Yussefiho algoritmus funguje
- Môžeme to eventuálne kombinovať. Vždy keď cyklus v hypergrafe spôsobuje selekčná hyperhrana. Eliminovať túto v štýle Wong Yussefiho algoritmu. Ak sa touto elimináciou hypergraf rozpadne na viac komponentov súvislosti optimalizovať každú komponentu vzľášť.
- Môžeme ignorovať selekčné hrany optimalizovať len joiny a selekcie aplikovať, čo najhlbšie v strome.
- Nápadov je veľa, inžiniersky to aj funguje, ale nevieme dokázať „dobré vlastnosti“ algoritmov na redukovateľných grafoch.

Konjunktívne dotazy

Ján Šturc

Jar, 2013

Konjunktívne dotazy – conjunctive queries (CQ)

- Konjunktívny dotaz pozostáva z jediného datalogového pravidla, ktorého telo obsahuje iba EDB predikáty.
 - $p(X) \leftarrow r_1, \dots, r_n$
- Konjunktívny dotaz definuje zobrazenie EDB do relácie odpovede, ktorá je definovaná hlavou tohto pravidla.
- Algebraická forma: $\Pi_X(R_1 \bowtie \dots \bowtie R_n)$.
- SQL: **select** X **from** R_1, \dots, R_n
where <spájacie podmienky>

Neskôr ho rozšíríme aj o selekcie.

Containment (subsumcia, pohltenie)

- $Q_1 \subseteq Q_2$ práve vtedy, ak pre všetky db D , $Q_1(D) \subseteq Q_2(D)$.
- Príklad:
 - $Q_1: p(x,y) \leftarrow \text{arc}(x,z), \text{arc}(z,y)$
 - $Q_2: p(x,y) \leftarrow \text{arc}(x,z), \text{arc}(w,y)$
- Databáza je o hranách grafu;
 - Q_1 definuje cesty dĺžky 2.
 - Q_2 definuje dvojice vrcholov také, že z prvého vychádza hrana a do druhého vchádza hrana.
- Vždy, keď existuje cesta z x do y , musí z x vychádzať hrana a do y vchádzať hrana. Preto každá dvojica $\langle x,y \rangle$, ktorá je odpoveď na dotaz Q_1 je aj odpoveď na dotaz Q_2 .
- Teda $Q_1 \subseteq Q_2$ alebo inak $Q_1 \Rightarrow Q_2$.

Dôvody zaoberať sa containmentom

- Optimalizácia: Ak sa nám podarí rozdeliť dotaz na konjunktívne dotazy, môžeme vynechať časti pohltené iným CQ.
- Integrácia IS: subsumcia CQ je často jediný spôsob ako zistiť, že nejaká informácia je zbytočná.
- Containment implikuje ekvivalenciu dotazov (programov):

$Q_1 \cong Q_2$ práve vtedy, keď $Q_1 \subseteq Q_2$ a $Q_2 \subseteq Q_1$.

- Akákoľvek teória programovania a optimalizácie potrebuje vedieť, kedy sú programy ekvivalentné.
- Zovšeobecnenie CQ o ktorom budeme hovoriť je najväčšia trieda dotazov, o ktorej vieme, že problém ekvivalencie je rozhodnuteľný.
- Umelá inteligencia v rôznych aplikáciach prijala logiku CQ, hoci CQ pochádzajú z teórie databáz.

Testovanie pohltenia

- Dva prístupy:
 1. Pohlcujúce zobrazenia.
 2. Kánonické databázy.
- Pre jednoduché CQ o akých sme hovorili doteraz je jedno, ktorý prístup použijeme.
- Containment je NP- complete, ale CQ sú obvykle dosť malé (krátke), tak nám ani exponenciálna zložitosť veľmi neuškodí.

Containment mapping – pohlcujuce zobrazenie

Definícia: Zobrazenie z premenných CQ Q_2 do premenných CQ Q_1 také, že:

1. hlava Q_2 sa zobrazí na hlavu Q_1
 2. a každý podcieľ Q_2 sa zobrazí do nejakého podcieľa Q_1 s tým istým predikátom,
- sa nazýva pohlcujuce zobrazenie. Hovoríme, že Q_2 pohlcuje Q_1 alebo, že Q_1 je obsiahnuté v Q_2 .

Pohlcujuce zobrazenie je „písmenkový“ homomorfizmus.

Písmenkový homomorfizmus môže mapovať premenné na konštanty, nie však „pôvodné“ konštanty na premenné.

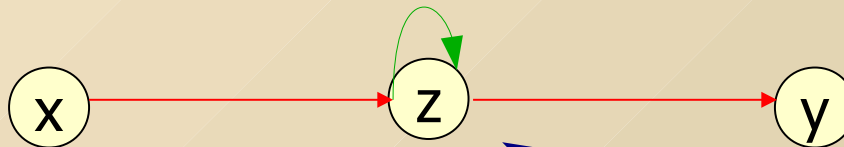
Veta: $Q_1 \subseteq Q_2$ práve vtedy, ak existuje pohlcujuce zobrazenie z Q_2 do Q_1 .

Príklad

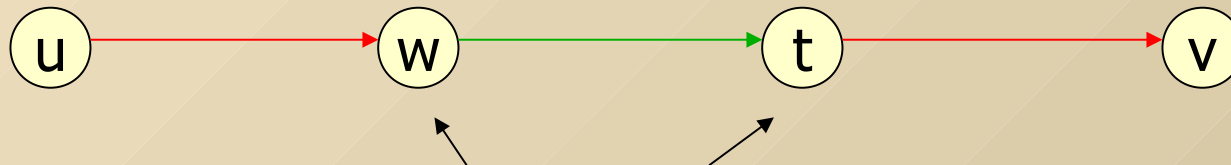
$Q_1: p(x, y) \leftarrow r(x, z), g(z, z), r(z, y)$

$Q_2: p(u, v) \leftarrow r(u, w), g(w, t), r(t, v)$

$Q_1:$



$Q_2:$



Pohlcajúce zobrazenie z Q_1 do Q_2 neexistuje, pretože existuje jediný podcieľ s g a z sa nemôže súčasne zobrazit' do w aj do t .

Pretože je možné $w=t$,
 $Q_1 \subseteq Q_2$.

Písmenkový homomorfizmus:

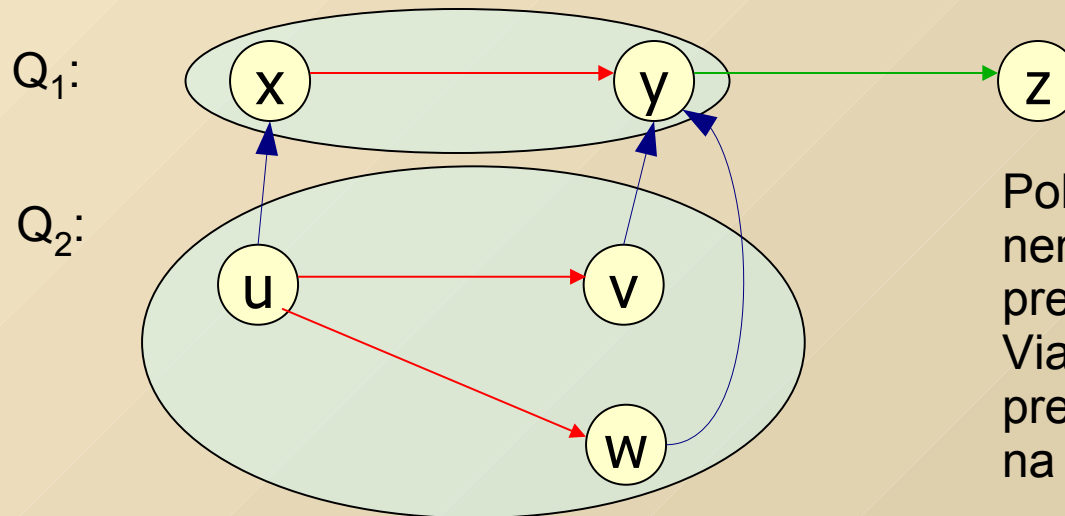
$u \rightarrow x, w \rightarrow z, t \rightarrow z, v \rightarrow y$

Iný príklad

$Q_1: p(x, y) \leftarrow r(x, y), g(y, z)$

$Q_2: p(u, v) \leftarrow r(u, v), r(u, w)$

Pohlcajúce zobrazenie z Q_1 do Q_2 neexistuje, pretože jediný podcieľ s g sa nemá do čoho zobrazit'.



Pohlcajúce zobrazenie nemusí pokryť všetky predikáty cieľového dotazu. Viac inštancií toho istého predikátu sa môže zobrazit' na jednu.

Písmenkový homomorfizmus:

$u \rightarrow x, v \rightarrow y, w \rightarrow y$

$Q_1 \subseteq Q_2$.

Dôkaz vety – $h: Q_2 \rightarrow Q_1 \Rightarrow Q_1(D) \subseteq Q_2(D)$

- Predpokladajme, že existuje pohlcujúce zobrazenie $h: Q_2 \rightarrow Q_1$.
- Musíme dokázať, že pre každú databázu D $Q_1(D) \subseteq Q_2(D)$.
- T.j. pre ľubovoľnú n -ticu $t \in Q_1(D)$ musíme dokázať, že platí $t \in Q_2(D)$.
- Čo znamená $t \in Q_1(D)$?
- Existuje substitúcia σ taká, že
 - pre každý podcieľ G z Q_1 je $G\sigma$ fakt v D
 - $t = H_1\sigma$, H_1 je hlava dotazu Q_1

Dokončenie prvej časti dôkazu

- Vypočítajme $(Q_2 h)\sigma = Q_2(\sigma \circ h)$

H_2 hlava Q_2

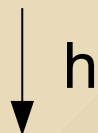


H_1 hlava Q_1

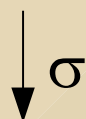


t

$G_{2,j}$ podcieľ Q_2



$G_{1,i}$ podcieľ Q_1



n -tica v D

Zložené

zobrazenie

$\sigma \circ h$ zobrazí každý
podcieľ Q_2 na
 n -ticu z D .

Dokazuje, že zložené zobrazenie $\sigma \circ h$ generuje odpovede na dotaz Q_2 a každá n -tica t je aj odpoveď na dotaz Q_1 t.j.

$$Q_1(D) \subseteq Q_2(D).$$

Petrifikované konjunktívne dotazy (Frozen CQ)

- Definícia:** *Petrifikovaný* konjunktívny dotaz *je databáza*, ktorá vznikne z dotazu Q nasledujúcimi krokmi:
- Pre každú premennú x dotazu Q vytvoríme unikátnu „premennú-konštantu“ x_p – *petrifikovanú premennú*.
1. Pre každý predikát v Q vytvoríme v databáze reláciu s rovnakým názvom.
 2. Pre každý podcieľ Q pridáme do relácie zodpovedajúcej tomuto podcieľu n-ticu, ktorá vznikne nahradením premenných v tomto podcieli príslušnými konštantami.

Príklad: $Q \equiv p(x, y) \leftarrow r(x, z), g(z, z), r(z, y)$
 $Q_p \equiv \{ r = \{(x_p, z_p), (z_p, y_p)\}, g = \{(z_p, z_p)\} \}$

Dôkaz vety 2. časť

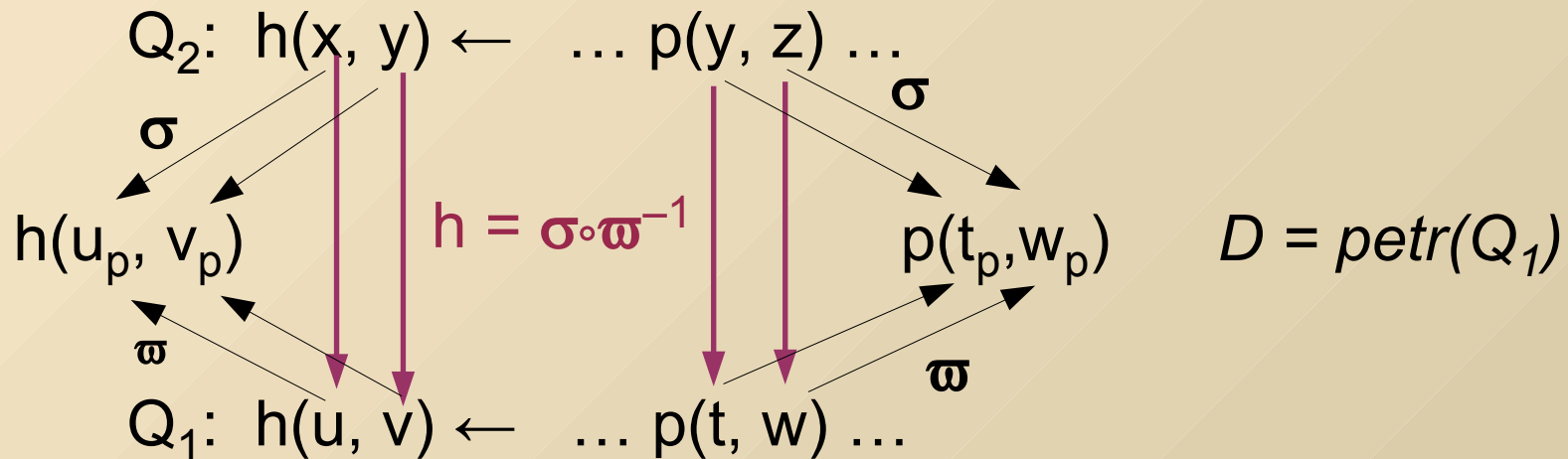
$$Q_1(D) \subseteq Q_2(D) \Rightarrow \exists h(h: Q_2 \rightarrow Q_1)$$

- Predpokladáme $Q_1(D) \subseteq Q_2(D)$ a musíme **dokázať**, že existuje pohlcujúce zobrazenie $h: Q_2 \rightarrow Q_1$.
- Nech D je petrifikácia dotazu Q_1 . (**$D = \text{petr}(Q)$**)
- Tvrdenie: $Q_1(D)$ obsahuje petrifikovanú hlavu Q_1 . T.j. hlavu Q_1 , v ktorej premenné sú nahradené konštantami – svojimi petrifikátmi.
 - Dôkaz: Základom petrifikácie je substitúcia $\varpi = \{x \mapsto x_p\}_{x \in \text{var}(Q)}$. Táto substitúcia vytvára relácie v D z podcieľov Q_1 . Ak použijeme ϖ na hlavu dostaneme **petrifikovanú hlavu**, preto musí byť táto odpoveď na $Q_1(D)$.

Dôkaz vety 2. časť – dokončenie

- Pretože $Q_1(D) \subseteq Q_2(D)$, petrifikovaná hlava Q_1 musí byť v $Q_2(D)$.
- Preto, existuje zobrazenie σ z premenných Q_2 do D také, že zobrazuje podciele Q_2 na n -tice D a hlavu Q_2 na petrifikovanú hlavu Q_1 .
- Ale n -tice D sú petrifikované podciele Q_1 , teda σ nasledované „depetrifikáciou“ ϖ^{-1} (Toto inverzné zobrazenie existuje, lebo petrifikáty premenných sú unikátne konštanty.) je pohlcujúce zobrazenie z Q_2 to Q_1 .
- $h = \sigma \circ \varpi^{-1}$.

Grafické znázornenie



$h = \sigma \circ \omega^{-1}$ zobrazí každý podcieľ $p(y, z)$ dotazu Q_2 do podcieľu $p(t, w)$ dotazu Q_1 a hlavu $h(x, y)$ dotazu Q_2 na hlavu $h(u, v)$ dotazu Q_1 .

Duálny pohľad na pohlcujúce zobrazenie

- Namiesto toho aby sme pohlcujúce zobrazenie považovali funkciu z premenných do „unikátnych konštánt“ budeme ho považovať za zobrazenie (injekciu) z atómov do atómov.
- Požiadavky na takéto zobrazenie:
 1. Hlava sa musí zobrazíť na hlavu.
 2. Každý podcieľ sa zobrazí na nejaký podcieľ.
 3. Z definície vyplýva, že žiadná premenná sa nemôže zobrazíť do dvoch rôznych premenných.

Kánonické databázy

- Hlavná myšlienka: je rozhodnúť, či $Q_1 \subseteq Q_2$ otestovaním, že platí $Q_1(D_1) \subseteq Q_2(D_1), \dots, Q_1(D_n) \subseteq Q_2(D_n)$, pre vybrané databázy D_1, \dots, D_n , kánonické databázy.
- Pre jednoduché konjunktívne dotazy stačí jediná kánonická databáza $\text{petr}(Q_1)$.
- Pre všeobecnejšie dotazy môžeme potrebovať nejakú väčšiu množinu kánonických DB.

Prečo test pomocou kánonických DB funguje.

Veta: Nech $D = \text{petr}(Q_1)$, H_1 je hlava Q_1 a ϖ petrifikačné zobrazenie.

Potom $Q_1 \subseteq Q_2$ práve vtedy, keď $H_1\varpi \in Q_2(D)$.

Dôkaz:

$(Q_1 \subseteq Q_2 \Rightarrow H_1\varpi \in Q_2(D))$: Nepriamo. Predpokladajme $H_1\varpi \notin Q_2(D)$. Ale $Q_1(D)$ iste obsahuje $H_1\varpi$ (Je to jeho petrifikovaná hlava), teda Q_2 nepohlcuje Q_1 . Spor.

$(H_1\varpi \in Q_2(D) \Rightarrow Q_1 \subseteq Q_2)$: Keďže $H_1\varpi$ je odpoveď na Q_2 v databáze D , existuje substitúcia σ , ktorá zobrazuje H_2 hlavu Q_2 na $H_1\varpi$ a podciele tela Q_2 na podciele $D = \text{petr}(Q_1)$. Zobrazenie $h = \sigma \circ \varpi^{-1}$ je pohlcujúci homomorfizmus z Q_2 do Q_1 . Teda $Q_1 \subseteq Q_2$.

Konštanty

- Konjunktívne dotazy často obsahujú konštanty.
 - Tieto zodpovedajú selekcii na rovnosť v relačnej algebre.
- Konštanty nemenia pohľad na pohlcujúce zobrazenie a testovanie pohltienia. Zmena v definícii pohlcujúceho a petrifikačného zobrazenia je:
 - Premenná sa môže zobrazit' na premennú alebo len na jednu konštantu.
 - Konštanta sa môže zobrazit' iba sama na seba.
- Pohlcujúce a petrifikačné zobrazenia sa podobajú substitúciám.

Príklad

$$\begin{array}{l} Q_2: \quad p(x) \leftarrow e(x, y) \\ \quad \quad \uparrow \quad \quad \uparrow \quad \quad \downarrow \\ Q_1: \quad p(u) \leftarrow e(u, 13) \\ \quad \quad \downarrow \quad \quad \downarrow \quad \quad \downarrow \end{array}$$

Pohlcajúce zobrazenie $h: Q_2 \rightarrow Q_1 \equiv \{ x \mapsto u, y \mapsto 13 \}$.

Preto $Q_1 \subseteq Q_2$.

Pohlcajúce zobrazenie z Q_2 do Q_1 by muselo zobrazit' konštantu 13 do premennej y . To je zakázané, preto také zobrazenie neexistuje.

Testovanie pohltenia

- Testovanie pohltenia konjunktívnych dotazov je NP-úplné.
- Prípad, keď Q_1 (pohltený dotaz) neobsahuje viac ako dva podciele s rovnakým predikátom, sa dá riešiť v lineárnom čase Satrianoovým algoritmom
- Redukciou na 2SAT.
- Uvedieme jeho zjednodušenú verziu pracujúcu v kvadratickom čase.

Sariayaov algoritmus

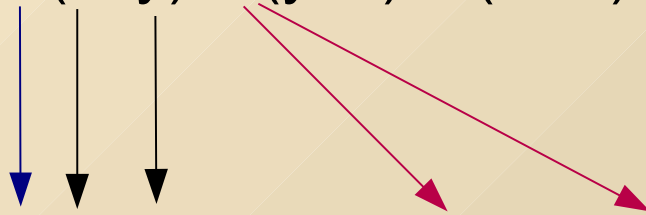
- Pre každý podcieľ $p(\dots) \in Q_2$, pre ktorý existuje jediný podcieľ s predikátom p v Q_1 , vieme presne na čo sa $p(\dots)$ zobrazí.
- Pre podcieľ Q_2 , pre ktorý existujú dva podciele v Q_1 , si vyberieme jeden z nich a odvodíme dôsledky.
- Môžu nastať dva prípady:
 - Odvodíme spor; Nejaká premenná „sa chce zobrazit“ na dve rôzne veci.
 - Backtrack; skúsime druhú alternatívu, ak neexistuje skončíme neúspechom.
 - Úspešne skončíme; Neexistujú ďalšie dôsledky a spor nevznikol.
 - Našli sme substitúcie za niektoré premenné Q_2 . Tieto si zapamätáme. Pokračujeme ďalším podcieľom, ak neexistuje, skončíme. Našli sme pohlcujúce zobrazenie.

Sariayaov algoritmus - „dôsledky“

- Ak sa $p(x_1, \dots, x_n)$ zobrazí na $p(y_1, \dots, y_n)$, potom vieme určiť na čo sa zobrazia premenné Q_2 .
V najhoršom unifikácii, ak sú to konjunktívne dotazy s termami.
- Ak $p(x_1, \dots, x_n)$ je podcieľ Q_2 , a premenná x_i sa zobrazuje na premennú z , a jediný podcieľ Q_1 s predikátom p obsahuje premennú z v i -tom argumente, potom sa $p(x_1, \dots, x_n)$ musí zobrazit' na tento podcieľ.

Príklad

$Q_2: p(x) \leftarrow a(x, y), b(y, z), b(z, w), a(w, x)$



$Q_1: p(v) \leftarrow a(u, v), a(v, u), b(u, t), b(t, v)$

Začneme výberom: $a(x, y) \rightarrow a(u, v)$.

Potom $x \mapsto u$ a $y \mapsto v$.

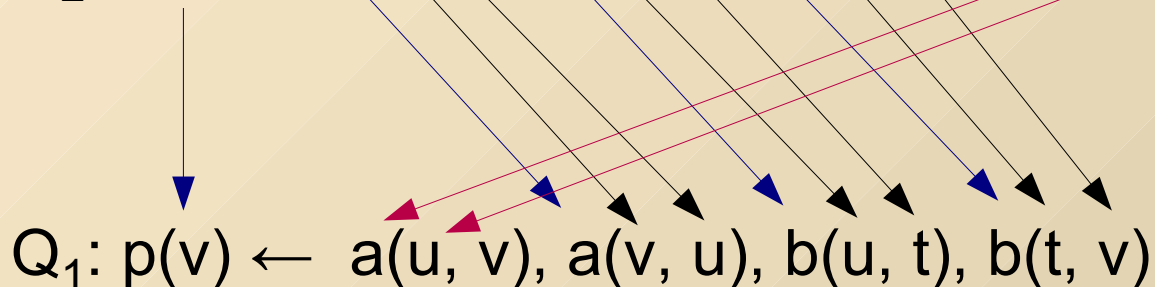
V ďalšom kroku sa musí $b(y, z)$ zobrazit' na nejaké $b(v, _)$.

Ale to neexistuje. Musíme backtracknúť a zmeniť prvý výber.

Keby sme začali hlavou urobili by sme lepšie. Môže to ušetriť zopár backtrackov.

Príklad – pokračovanie

$Q_2: p(x) \leftarrow a(x, y), b(y, z), b(z, w), a(w, x)$



Teraz už vieme, že každé pohlcujúce zobrazenie z Q_2 do Q_1 musí zobrazit' $a(x, y)$ na $a(v, u)$. Teda $x \mapsto v$ a $y \mapsto u$.

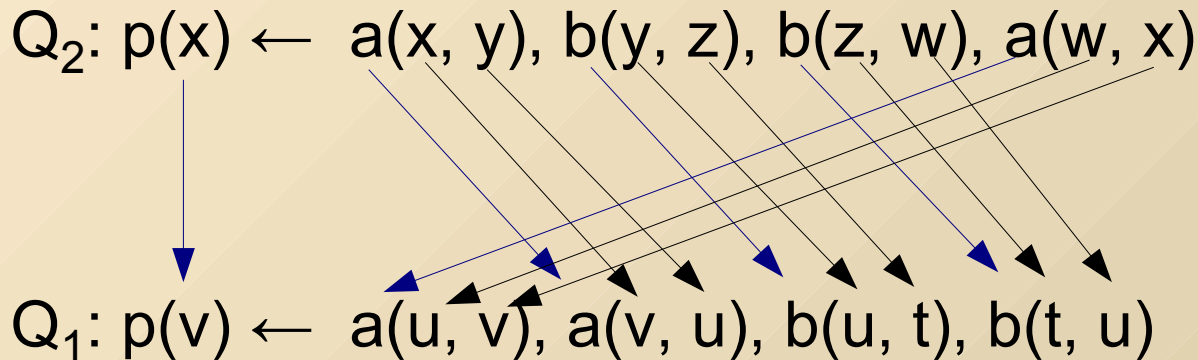
Následne $b(y, z)$ sa musí zobrazit' na $b(u, t)$ a $z \mapsto t$.

Pre ďalšie b je jediná možnosť $b(z, w)$ na $b(t, v)$, ktorá generuje $w \mapsto v$. Všetky premenné Q_2 sú už definované.

Ale posledné zobrazenie $a(w, x)$ na $a(u, v)$ zlyhá, lebo premenná w je už zobrazená na v .

Definitívne zlyhanie: záver $Q_1 \not\subseteq Q_2$.

Príklad – mierná modifikácia



Algoritmus prebiehá podobne ako v predošlom príklade. V predposlednom kroku sa w priradí u . Vzniklé zobrazenie $h = \{x \mapsto v, y \mapsto u, z \mapsto t, w \mapsto u\}$ je konzistentné so zobrazením posledného predikátu Q_2 na prvý predikát Q_1 . Žiadne ďalšie dôsledky sa negenerujú.

Zobrazenie h je pohlcujúci homomorfizmus. Teda $Q_1 \subseteq Q_2$.

Rozšírenia konjunktívnych dotazov

- Zjednotenie
- Aritmetika
- Negácie

Pohltenie pre zjednotenie CQ

Veta: $P_1 \cup \dots \cup P_k \subseteq Q_1 \cup \dots \cup Q_n$ práve vtedy, keď pre každé P_i existuje Q_j také, že $P_i \subseteq Q_j$.

Dôkaz:

$(\forall P_i \exists Q_j P_i \subseteq Q_j \Rightarrow P_1 \cup \dots \cup P_k \subseteq Q_1 \cup \dots \cup Q_n)$: Zrejmé.

Naopak. Predpokladáme $P_1 \cup \dots \cup P_k \subseteq Q_1 \cup \dots \cup Q_n$.

Nech D je petrifikovaný dotaz P_i . Pretože $P_i(D)$ obsahuje svoju petrifikovanú hlavu a platí $P_i \subseteq Q_1 \cup \dots \cup Q_n$, musí nejaké $Q_j(D)$ obsahovať petrifikovanú hlavu P_i .

Teda $P_i \subseteq Q_j$.

Pohltenie konjunktívnych dotazov datalógovým programom

- Nech Q je CQ a P datalógový program.
- Pre každú EDB databázu D aj P aj Q vráti reláciu, je teda zmysluplné sa pýtať, či $Q \subseteq P$. Presnejši, či platí $Q(D) \subseteq P(D)$ pre každé D .
- Test pohltenia
 - Nech D je kánonická DB pre Q .
 - Vypočítajte $P(D)$ a otestujte, či obsahuje petrifikovanú hlavu Q .
 - Ak áno, $Q \subseteq P$. V opačnom prípade D je protipríklad.

Príklad

$Q : p(x,y) \leftarrow a(x,z), a(z,w), a(w,y)$

$P : p(x,y) \leftarrow a(x,y)$

$p(x,y) \leftarrow p(x,z), p(z,y)$

- Intuitívne: Q = cesty dĺžky 3; P = všetky cesty.
- Petrifikované $Q : D = \{a(x_p, z_p), a(z_p, w_p), a(w_p, y_p)\}$.

$D = \{a(x_p, z_p), a(z_p, w_p), a(w_p, y_p)\}$

- Podľa prvého pravidla: $p(x_p, z_p), p(z_p, w_p), p(w_p, y_p)$
- Podľa druhého pravidla: $p(x_p, w_p), p(z_p, w_p); p(x_p, y_p)$
- Petrifikovaná hlava Q . Teda $Q \subseteq P$.

Iné subsumcie

- Je dvojite exponenciálny problém $O(2^{2^n})$ rozhodnúť, či konjunktívny dotaz pohltí datalogový program.
- Je nerozhodnuteľné, či datalogový program pohltí iný datalogový program.

Konjunktívne dotazy s negáciou

- Dovolíme aj negované podciele
- Príklad:

$Q_1: p(x,y) \leftarrow a(x,z), a(z,y), \neg a(x,y)$ cesta dĺžky práve dva

$Q_2: p(x,y) \leftarrow a(x,y), \neg a(y, x)$ hrana len jedným smerom

Levy-Sagivov test

- Testuje či $Q_1 \subseteq Q_2$:
 1. Ako množinu kánonických databáz $Can(Q_1)$ pre Q_1 uvažujeme množinu všetkých databáz s reláciami pre predikáty Q_1 , s doménou zo symbolov $1, \dots, n$, kde n je počet premenných dotazu Q_1 .
 2. Ak existuje $D \in Can(Q_1)$ také, že $Q_1(D) \not\subseteq Q_2(D)$.
Potom $Q_1 \not\subseteq Q_2$.
 3. V opačnom prípade $Q_1 \subseteq Q_2$.

Príklad

$Q_1: p(x,y) \leftarrow a(x,z), a(z,y), \neg a(x,y)$

$Q_2: p(x,y) \leftarrow a(x,y), \neg a(y, x)$

- Skúsime $D = \{a(1,2), a(2,3)\}$.
- $Q_1(D) = \{p(1,3)\}$.
- $Q_2(D) = \{p(1,2), p(2,3)\}$.
- Teda $Q_1 \not\subseteq Q_2$.

Intuícia

- Nestačí uvažovať len petrifikované telo dotazu Q_1 .
- Dôvod je, že v prípade negácie subsumcia môže byť narušená aj tým, že niektorým premenným je priradená tá istá konštanta.

Konjunktívne dotazy so zabudovanými predikátmi

- Špeciálny prípad: aritmetické predikáty usporiadania ako napr. $<$.
 - Úplné usporiadanie na doménach.
 - Coddova relačná algebra
- Všeobecný prípad: predikáty, ktoré majú pevný matematický význam, ale môžu byť zložitejšie ako jednoduché aritmetické porovnanie.
 - Príklady:
 - aritmetika, sčítanie, násobenie, ...
 - množinové premenné a množinové inklúzie
 - operácie s reťazcami a dátumami

Konjunktívne dotazy s predikátom $<$

- Na testovanie $Q_1 \subseteq Q_2$, uvažujeme kánonické bázy dát – všetky databázy vytvorené z obyčajných (nie aritmetických) podcieľov Q_1 , nad doménou $1, \dots, n$. Kde n je počet premených v Q_1 .
- Ekvivalentné je: rozdeliť premenné Q_1 do blokov a usporiadať bloky podľa relácie $<$.

Príklad

$Q_1: p(x,z) \leftarrow a(x,y), a(y,z), x < y$

$Q_2: p(u,w) \leftarrow a(u,v), a(v,w), u < w$

- Existuje 13 usporiadaných skupín:
 - 6 usporiadaní pre skupiny po jednom $\{x\}\{y\}\{z\}$.
 - 3×2 usporiadaní pre tri 2-1 skupiny, ako $\{X\}\{Y,Z\}$.
 - 1 jediná skupina po troch $\{X,Y,Z\}$.
- Zvolme skupinu: $\{x,z\}\{y\}$; nech napr. $x = z = 1$ and $y = 2$.
- Potom sa telo Q_1 transformuje na $D = \{a(1,2), a(2,1)\}$, a $x < y$ platí, teda hlava $p(1,1)$ je v $Q_1(D)$.

Príklad - dokončenie

$Q_2: p(u,w) \leftarrow a(u,v), a(v,w), u < w$

$D = \{a(1,2), a(2,1)\}$

- Tvrdenie $Q_2(D) = \emptyset$, pretože existujú dva spôsoby ako splniť prvé dva podciele
 - $u = w = 1$ a $v = 2$, alebo
 - $u = w = 2$ a $v = 1$.
- V oboch prípadoch, $u < w$ neplatí.
- Teda $Q_1 \not\subseteq Q_2$.

Aritmetika všeličo kazí

- Veta o zjednotení CQ neplatí.
- Príklad:
 $P : p(x) \leftarrow a(x), 10 \leq x, x \leq 20$
 $Q : p(x) \leftarrow a(x), 10 \leq x, x \leq 15$
 $R : p(x) \leftarrow a(x), 15 \leq x, x \leq 20$
 $P \subseteq Q \cup R$, ale ani $P \subseteq Q$ ani $P \subseteq R$ neplatí.
- Veta o pohlcujúcom homomorfizme neplatí.
- Príklad:
 $Q_1: \text{panic} \leftarrow a(x,y), a(y,x)$
 $Q_2: \text{panic} \leftarrow a(u,v), u \leq v$

Q_1 je cyklus dĺžky 2, Q_2 neklesajúca hrana. Očakavali by sme $Q_1 \subseteq Q_2$, ale zabudovaný predikát \leq sa nemá na čo zobrazit'.

Pohlcajúci homomorfizmus pre zabudované predikáty

- „Rektifikované“ pravidlá a normálna forma pre CQ s zabudovanými predikátmi.
- Varianta vety o pohlcajúcom homomorfizme platí aj pre zabudované predikáty. Táto veta platí aj pre iné predikáty ako aritmetické porovnania, ale na rektifikáciu potrebujeme aspoň rovnosť „=“.
- Rektifikácia:
 1. Žiadná premenná sa nesmie medzi argumentami obyčajných predikátov tela a argumentami hlavy vyskytnúť viac než raz.
 2. V hlave ani v obyčajných predikátoch tela sa nesmia vyskytovať konštanty.

Pravidlá pre rektifikáciu

- Zavedieme nové premenné namiesto násobne sa vyskytujúcich premenných a konštánt.
- Nové premenné „previažeme“ s pôvodnými a konštantami pomocou predikátu rovnosti.
- Príklady:

$\text{panic} \leftarrow a(x,y), a(y,x)$ sa transformuje na

$\text{panic} \leftarrow a(x,y), a(u,v), u=y, v=x$

$p(x) \leftarrow q(x,y,x), r(y,a)$ sa transformuje na

$p(z) \leftarrow q(x,y,w), r(v,u), x=w, x=z, y=v, u=a$

Gupta-Zhang-Ozsoyogluov test

- Nech Q_1 a Q_2 sú rektifikované pravidlá.
- Nech M je množina všetkých pohlcujúcich homomorfizmov z obyčajných (nezabudovaných) podcieľov Q_2 do obyčajných podcieľov Q_1 . Pre rektifikované pravidlá je každé zobrazenie podcieľov do podcieľov s rovnakými predikátmi pohlcujúci homomorfizmus.

Veta: $Q_1 \subseteq Q_2$ práve vtedy, keď zo zabudovaných podcieľov Q_1 logicky vyplýva zjednotenie cez všetky h z M h -obrazov zabudovaných podcieľov Q_2 .

Príklad

$Q_2: \text{panic} \leftarrow a(u,v), u \leq v$



$Q_1: \text{panic} \leftarrow a(x,y), a(z,w), z=y, w=x$

$M = \{h_1 = \{u \mapsto x, v \mapsto y\}, h_2 = \{u \mapsto z, v \mapsto w\}\}$

$(u \leq v)[h_1] = x \leq y$

$(u \leq v)[h_2] = y \leq x$

Treba ukázať $((z=y) \wedge (w=x)) \Rightarrow ((x \leq y) \vee (z \leq w))$

Dôkaz:

$(w \leq z) \vee (z \leq w)$ veta o úplnom usporiadaní

$(x \leq y) \vee (z \leq w)$ v prvej zátvorke sme nahradili rovné rovným

Axiomy pre nerovnosti

1. $x \leq x$.
2. $x < y$ implikuje $x \leq y$.
3. $x < y$ implikuje $x \neq y$.
4. $x \leq y$ a $x \neq y$ implikuje $x < y$.
5. $x \neq y$ implikuje $y \neq x$.
6. $x < y$ a $y < z$ implikuje $x < z$.
7. $x \leq y$ a $x \leq y$ implikuje $x \leq z$.
8. $x \leq z$, $z \leq y$, $x \leq w$, $w \leq y$ a $w \neq z$ implikuje $x \neq y$.

Veta: Systém axiém 1 – 8 je zdravý a úplný.

Dôkaz: Je to netriviálna veta z elementárnej aritmetiky.
Pozrite do knihy.

Minimalizácia konjunktívnych dotazov

Ján Šturc
Jar, 2013

Motivačný príklad 1 – fp

- Funny path
$$r_1: \text{fp}(x, y) \leftarrow e(x, y)$$
$$r_2: \text{fp}(x, y) \leftarrow e(y, x), \text{fp}(z, x)$$
- Po rozvinutí rekurzie z pravidla r_2 dostaneme
 - $r_{21}: \text{fp}(x, y) \leftarrow e(y, x), e(z_1, x)$
$$\cdot \cdot \cdot$$
$$r_{2k}: \text{fp}(x, y) \leftarrow e(y, x), e(z_1, x), e(z_2, x), \dots, e(z_k, x)$$
$$\cdot \cdot \cdot$$
 - $r_{2k} \subseteq r_{21}$ prvé dva podciele sa mapujú identicky, na ďalšie sa nemapuje nič.
 - $r_{21} \subseteq r_{2k}$ pre každé $i > 1$, z_i sa mapuje na z_1 .
- Všetky rozvoje sú ekvivalentné a druhé pravidlo môžeme nahradiť nerekurzívnym pravidlom:
$$r_2': \text{fp}(x, y) \leftarrow e(y, x), e(z, x)$$

Motivačný príklad 2 – EDM

- Daná je extenzionálna databáza $ed(z,o)$, $dm(o,v)$ a view $edm = ed \bowtie dm$. Dotaz $Q_1 = \Pi_o \sigma_{z='m'} edm$. Otázka je, či môžeme tento dotaz optimalizovať na $Q_2 = \Pi_o \sigma_{z='m'} ed$?
- Preklad do rektifikovaného datalógu:
 $Q_1: a(o) \leftarrow ed(z, o), dm(o, v), z = 'm'$
 $Q_2: a(o) \leftarrow ed(z, o), z = 'm'$
- Zrejme $Q_1 \subseteq Q_2$. Opačné pohltenie neplatí.
- $Q_1 \not\subseteq Q_2$. Ale v prípade, že ed a dm sa spája bezstrátovo t.j. relácie ed a dm vznikli normalizáciou relácie edm napr. vzhľadom na funkčné závislosti $o \leftrightarrow v$ (oddelenie má práve jedného vedúceho), dávajú oba dotazy „vždy“ tú istú odpoveď.

Slabé pohltenie – weak containment

- Pohltenie bolo definované, že pre každú databázu D platí $Q_1(D) \subseteq Q_2(D)$.
- Oslabíme požiadavku pre každú len na databázy, ktoré sa spájajú bezstrátovo.

Definícia: Nech Q_1 a Q_2 sú dva dotazy na databázu s relačnou schémou R_1, \dots, R_n . Hovoríme, že Q_2 slabo pohlcuje Q_1 , ak pre každú databázu D so schémou R_1, \dots, R_n takú, že splňuje podmienku $\forall i (R_i = \prod_{R_i} (R_1 \bowtie \dots \bowtie R_n))$ platí $Q_1(D) \subseteq Q_2(D)$. Píšeme $Q_1 \subseteq_w Q_2$.

- Predpoklad $\forall i (R_i = \prod_{R_i} (R_1 \bowtie \dots \bowtie R_n))$ bezstrátovosti spojenia relácií D sa nazýva aj predpokladom „univerzálnej relácie / reprezentatívnej inštancie“. Tiež globálna konzistencia. **Vzťahuje sa aj na podciele dotazu.**

Poznámka

- Predpoklad „univerzálnej relácie“ je o schéme a dotazoch
 - Môžeme predpokladať že schéma vznikla normalizáciou (dekompozíciou) jedinej relácie
 - a dotazy sa týkajú len „spätných“ spojení dekomponovaných relácii podľa rovnako nazvaných atribútov.
- Relácie sú jednoznačne určené množinou svojich atribútov, ich mená sú zbytočné. Presne ako sme to robili pri normalizácii v minulom semestri.

Slabá ekvivalencia dotazov

Definícia: Hovoríme, že dotazy Q_1 a Q_2 sú slabo ekvivalentné, (Píšeme $Q_1 \cong_w Q_2$.) práve vtedy, ak $Q_1 \subseteq_w Q_2$ a $Q_2 \subseteq_w Q_1$.

- Teoreticky je možnosť pojem slabého pohltenia a slabej ekvivalencie zovšeobecniť pre ľubovoľnú množinu \mathcal{C} podmienok (constraints) na databázy.
Píšeme $Q_1 \subseteq_c Q_2$ a $Q_1 \cong_c Q_2$.
- Zrejme pohltenie, ekvivalencia implikuje slabé pohltenie, slabú ekvivalenciu.

Tablá – tableaux

- Tabló (tableau) je rektangulárne (obdĺžníkovité) pole.
- Jeho stĺpce zodpovedajú atribútom („oblúbeným“ premenným), tieto sú zapísané v hlavičke (prvom riadku)
- Druhý (**posledný**) riadok (sumár) obsahuje hlavu dotazu. Premenné vyskytujúce sa v tomto riadku nazývame významné (distinguished). Zvyšné premenné sa nazývajú nevýznamné (nondistinguished).
- Riadky tabla zodpovedajú jednotlivým podcieľom dotazu.
- Tabló je množina riadkov (idempotencia \wedge)
- Konvencia: nevýznamné premenné, ktoré sa v table (dotaze) vyskytujú iba raz (v datalógu podčiarkovník) možno nahradiť prazdným symbolom (blank).

Konštrukcia tabla k výrazu relačnej algebry

1. Výraz:	Tabló:
$R(x_1, \dots, x_n)$	$\frac{x_1 \dots x_n}{x_1 \dots x_n}$
	$x_1 \dots x_n$

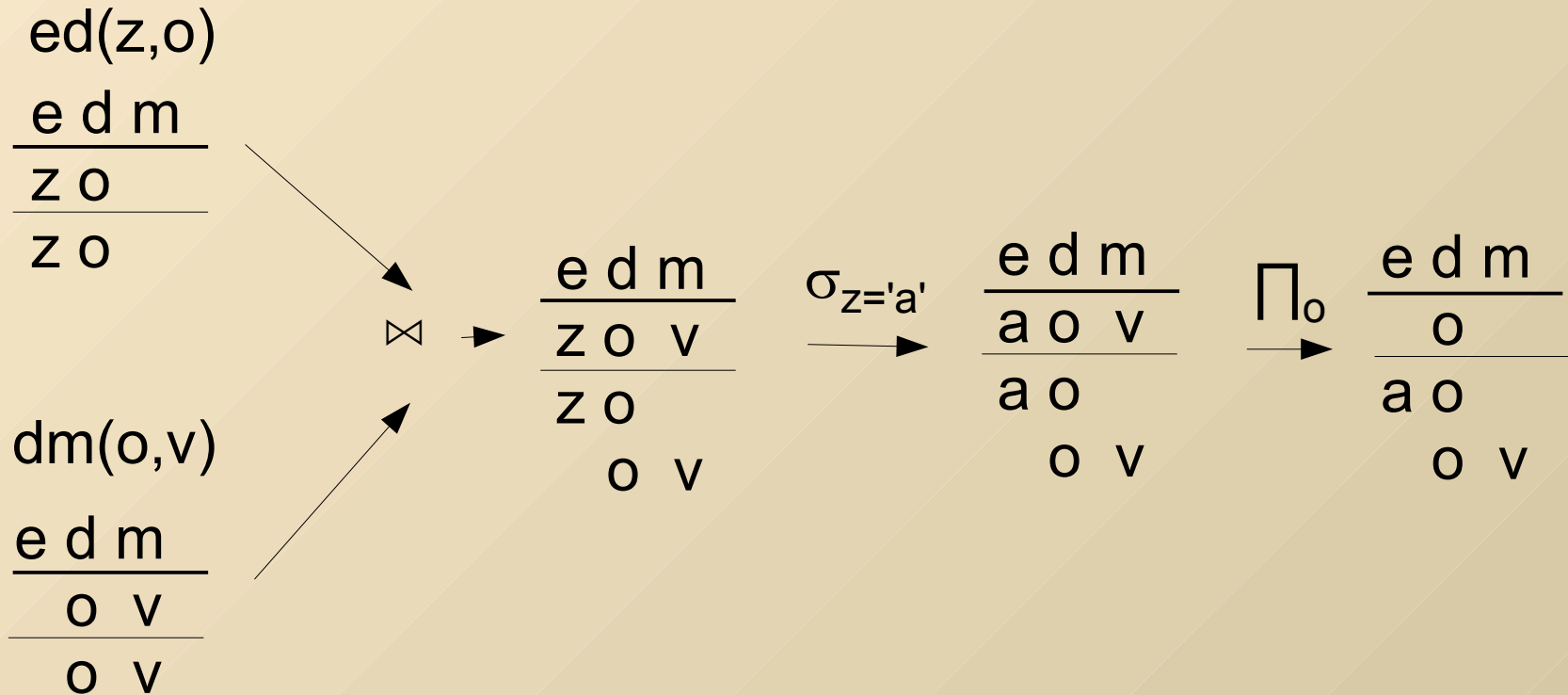
2. Ak T je tabló pre výraz $E(x_1, \dots, x_n)$. Potom tabló pre $\sigma_{x_i=a}E$ dostaneme z tabla T tak, že v table T nahradíme všetky výskyty premennej x_i konštantou a .
3. Ak T je tabló pre výraz $E(x_1, \dots, x_n)$. Potom tabló pre $\sigma_{x_i=x_j}E$ dostaneme z tabla T tak, že v table T stotožníme všetky výskyty premennej x_i a x_j .

Pozn.: Operácie 2, 3, sa vykonajú počnúc 2. riadkom. Hlavička zostane nezmenená.

Konštrukcia tabla k výrazu relačnej algebry – pokračovanie

4. Ak T je tabló pre výraz $E(x_1, \dots, x_n)$. Potom tabló pre $\prod_{x_{i_1} \dots x_{i_k}} E$, dostaneme z tabla T tak, že v sumári T premenné x_{i_1}, \dots, x_{i_k} nahradíme prázdnyimi znakmi. Vo zvyšných riadkoch zostanú, ale stanú sa nevýznamné.
5. Ak T je tabló pre $E(\mathbf{x}, \mathbf{y})$ a S je tabló pre $F(\mathbf{y}, \mathbf{z})$. Potom tabló pre $E(\mathbf{x}, \mathbf{y}) \bowtie F(\mathbf{y}, \mathbf{z})$ dostaneme z tabiel' T a S tak, že zjednotíme ich riadky a „unifikujeme“ ich sumáre.
 - Priorita: konštanta, premenná, blank
 - Unifikačnú substitúciu aplikujeme na všetky riadky
 - Ak sa nedajú unifikovať je sumár prázdny. Tabló mapuje každú EDB do prázdnej relácie. To nastane napr., ak sumáre majú na danej pozícii rôzne konštanty.

Príklad $\Pi_o \sigma_{z='a'} edm$



Tablá a petrifikované dotazy

- Tabló je grafické znázornenie petrifikovaného dotazu.
 - Sumár je petrifikovaná hlava
 - Riadky sú petrifikované podciele tela
- Premenné v table zodpovedajú petrifikovaným premenným. Rozlíšenie medzi významnými a nevýznamnými premennými je informácia navyše.
- Pohlcujúci homomorfizmus medzi tablami je pohlcujúci homomorfizmus medzi petrifikovanými dotazmi.
- Ekvivalencia tabiel je ekvivalencia petrifikovaných dotazov.
- Predpoklad „univerzálnej relácie“ implikuje, že pohlcujúci homomorfizmus nemusí „strážiť“ predikáty podcieľov, ale len ich premenné (atribúty).

Minimalizácia tabiel 1

- Veta o pohlcujúcom homomorfizme platí aj pre tablá. Pri konštrukcii pohlcujúceho homomorfizmu sa **význačné premenné** môžu zobrazit' podobne ako konštanty **len samé na seba**.
- $T_1 \subseteq_w T_2$ práve vtedy, keď existuje pohlcujúci homomorfizmus η z T_2 do T_1 .

Dôsledok: Nech r_1 a r_2 sú dva riadky tabla T , η je pohlcujúci homomorfizmus z riadku r_2 na riadok r_1 , potom pre tabló $T' = T_\eta - \{r_1\}$ platí: $T' \subseteq_w T$.

Dôkaz: Zrejmé. Pohlcujúce zobrazenie riadku na riadok je substitúcia. Pohlcujúci homomorfizmus celého tabla dostane zložením identickej substitúcie a η .

Minimalizácia tabiel 2

- Obrátené tvrdenie vo všeobecnosti neplatí. Ale platí

Veta: Nech Q je konjunktívny dotaz. Potom existuje konjunktívny dotaz Q' , ktorého množina podcieľov je podmnožina množiny podcieľov Q a je minimálna.

Dôkaz: Nech T je tabló pre Q a t jeho sumár. Vyberieme riadok r a otestujeme, či existuje homomorfizmus φ taký, že $T_\varphi \subseteq_w T - \{r\}$ a $s_\varphi = s$. Homomorfizmus môže zobrazovať nevýznamné premenné na iné premenné alebo konštanty, vyskytujúce sa v riadku r .

Jednoduchý príklad

$$fp(x,y) \leftarrow e(y, x), e(z_1, x), e(z_2, x), \dots, e(z_k, x)$$

T:

	Z	DO
	x	y
1	y	x
2	z_1	x
3	z_2	x
...		
k	z_k	x

Postupnými substitúciami $\varphi_i = [z_i \mapsto z_1]$ eliminujeme riadky 3, ..., k. Dostaneme minimalizované tabló T' .

$$T' \subseteq_w T; \quad \varphi = \{[z_i \mapsto z_1]\}_{i=2}^k$$

$$T \subseteq_w T'; \quad \iota = \text{identická subst.}$$

$$\text{Teda } T' \cong_w T.$$

Ekvivalentné minimálne tabló.

T':

	Z	DO
	x	y
1	y	x
2	z_1	x

Iný príklad

R	A	B
	x	x
	x	y_1
	y_1	y_2
	\vdots	\vdots
	y_{n-1}	y_n
	y_n	x
	x	

q_1

R	A	B
	x	x
	x	

q_2

$q_1 \cong q_2$

Žiadná lokálna optimalizácia
netransformuje q_1 na q_2 .

Francúzi a nemci píšu sumár na spodok tabla.

Mali by sme tiež. Je to účtovnícky zvyk.

Iný príklad – edm

$$T: \begin{array}{c} \frac{e \quad d \quad m}{o} \\ a \quad o \quad v_1 \\ e_1 \quad o \quad v \end{array}$$

$$T': \begin{array}{c} \frac{e \quad d \quad m}{o} \\ a \quad o \quad v_1 \end{array}$$

$$\varphi = [e_1 \mapsto a, v \mapsto v_1]; \quad T' \subseteq_w T$$

$$\iota = \text{identická subst.}; \quad T \subseteq_w T'$$

$$\text{Znovu } T' \cong_w T.$$

Napriek tomu tablá T' a T nereprezentujú logicky ekvivalentné dotazy, lebo riadky 1 a 2 zodpovedajú rôznym predikátom.

Dotaz T je ekvivalený tvrdeniu:

$$T' \wedge (\exists v) dm(o, v).$$

V dôsledku bezstrátovosti spojenia $ed \bowtie dm$ to naozaj platí.

Ešte iný príklad

$$q = \pi_{AB}(\sigma_{B=5}(R)) \bowtie \pi_{BC}(\pi_{AB}(R) \bowtie \pi_{AC}(\sigma_{B=5}(R)))$$

$$(\mathbf{T}, u) =$$

R	A	B	C
	x	5	z_1
	x_1	5	z_2
	x_1	5	z
u	x	5	z

$$(\mathbf{T}', u) =$$

R	A	B	C
	x	5	z_1
	x_1	5	z
u	x	5	z

$$q' = \pi_{AB}(\sigma_{B=5}(R)) \bowtie \pi_{BC}(\sigma_{B=5}(R))$$

Minimalizácia tabiel

- Predošlé tvrdenie umožňuje minimalizovať tablá postupným vynechávaním riadkov.
- Dokonca stačí hľadať riadok, ktorý subsumuje (pohlcuje) iný riadok (term matching). $O(m^2n)$, kde m je počet riadkov a n dĺžka riadku.
- Pri bližšom pohľade sa tablá podobajú na petrifikované dotazy. Sumár zodpovedá petrifikovanej hlave a riadky tabla zodpovedajú petrifikovanému telu.
- Petrifikovať môžeme každý dotaz.
- Formálne môžeme aj ku každému dotazu zostaviť tabló, musíme však pre všetky argumenty všetkých podcieľov vytvoriť stĺpec. Je však otazné nakoľko pre takéto dotazy platí predpoklad „univerzálnej relácie“.

Uzatváracia procedúra – chase

- Môžeme využiť, že databáza splňuje nejaké podmienky (constraints) (napr.)
 - funkčné závislosti (fd)
 - multizávislosti (mvd)
 - spojovacie závislosti (jd)
 - inklúzne závislosti (ind)
- Ako? Pred alebo súčasne s minimalizáciou vynutíme, aby tabló splňovalo podmienky.

Vynutenie fd

- Nech $\mathbf{A} \rightarrow B$, kde \mathbf{A} je množina atribútov a B atribút
- Vynutenie tejto závislosti na table T , znamená, že pre každé dva riadky r_1, r_2 platí, ak sa zhodujú v atribútoch (stĺpcoch) \mathbf{A} musia sa zhodovať aj v stĺpci B .
- Vyberieme riadky r_1, r_2 otestujeme rovnosť $r_1.\mathbf{A} = r_2.\mathbf{A}$
- Pre stĺpec B sú nasledujúce možnosti
 - $r_1.B = r_2.B$, netreba nič robiť fd je splnená.
 - $r_1.B \neq r_2.B$ symboly sú rôzne konštanty. Nemôže nastať je to kontrapríklad proti fd, ktorá má platiť.
 - Riadky majú v pozícii B rôzne symboly. Stotožníme ich. Vyberieme jeden z nich, preferujeme najprv konštantu, potom významnú premennú. Všetky výskyty druhej premennej v table T nahradíme vybraným symbolom.

Zložitosť vynútenia fd

- Algoritmus končí, lebo každou aplikáciu fd ubudne jeden symbol z tabla.
- Nech m je počet riadkov, n počet atribútov a f počet funkčných závislostí.
- Existuje $\binom{m}{2}$ dvojíc riadkov
- Test nájdenie **A** a porovnanie B je zložitosti $O(n)$.
- Pre nahradenie nezhody v najhoršom preskanujeme celé tabló $O(mn)$
- A máme f závislostí
- Celková zložitosť sa dá odhadnúť $O(fm^3n^2)$
- Možno to nie je presné, ale v každom prípade polynomiálna zložitosť.

Vynutenie mvd a jd tuple generating dependencies

- V schéme $R(\mathbf{x}, \mathbf{y}, \mathbf{z})$ mvd $\mathbf{x} \rightarrow \mathbf{y}$ znamená:
 $R(\mathbf{x}, \mathbf{y}_1, \mathbf{z}_1) \wedge R(\mathbf{x}, \mathbf{y}_2, \mathbf{z}_2) \Rightarrow R(\mathbf{x}, \mathbf{y}_1, \mathbf{z}_2) \wedge R(\mathbf{x}, \mathbf{y}_2, \mathbf{z}_1)$
- V tabulárnej forme to znamená doplnenie riadkov
- Podobne jd $\bowtie R_1, \dots, R_n$ znamená, že ak existujú riadky pre jednotlivé relácie, existuje aj riadok pre ich spojenie

<u>A</u>	<u>B</u>	<u>C</u>
x	y₁	z₁
x	y₂	z₂
x	y₁	z₂
x	y₂	z₁

Uvedená mvd je to isté ako jd

$\bowtie \mathbf{AB}, \mathbf{BC}$

Znamená to, že schémy sa spájajú bezstrátovo.

Vynútením mvd a jd môže počet riadkov v table exponenciálne rásť.

Vnorené závislosti – embedded dependencies

- Mvd a jd môžu byť aj vnorené znamená to, že sa nevzťahujú na celú množinu atribútov tabla, ale len na nejakú jej podmnožinu
- V schéme $R(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{v})$ mvd $\mathbf{x} \twoheadrightarrow \mathbf{y} | \mathbf{z}$ znamená: $\exists(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$
 $(R(\mathbf{x}, \mathbf{y}_1, \mathbf{z}_1, \mathbf{v}_1) \wedge R(\mathbf{x}, \mathbf{y}_2, \mathbf{z}_2, \mathbf{v}_2) \Rightarrow R(\mathbf{x}, \mathbf{y}_1, \mathbf{z}_2, \mathbf{v}_3))$

Príklad:

A	B	C	D
x	y₁	z₁	v₁
x	y₂	z₂	v₂
x	y₁	z₂	v₃
x	y₂	z₁	v₄

Vynutenie ind

- Inklúzna závislosť znamená $\exists \mathbf{y} R(\mathbf{x}, \mathbf{y}) \Rightarrow \exists \mathbf{z} S(\mathbf{x}, \mathbf{z})$.
Ničmenej premenné \mathbf{x} sa môžu v R a S vyskytovať na pozícií rôznych atribútov.
- Inklúzna závislosť eventuálne generuje nejaké riadky do tabla.
- Ind v kombinácii s fd alebo mvd môžu spôsobiť zacyklenie tabló bude stále rásť. Algoritmus vynutenia neskončí. Nerozhodnuteľnosť.

„Silná“ ekvivalencia

Príklad:

$$p(x,y) \leftarrow q(x,y), s(u,v)$$

Tabló bez hlavičky

x	y
x	y
	u v

x	y	
x	y	
	u	v
		q
		s

Po doplnení anonýmnych premenných a minimalizácii. Prvý riadok pohltí druhý. Dostaneme slabo ekvivalentný dotaz: $p(x,y) \leftarrow q(x,y)$
Ekvivalencia neplatí ak $s(u,v)$ je prázdne.

Aby sme zabránili nesprávnemu stotožneniu riadkov pridáme ku každému riadku tag – reláciu (pociel'), z ktorého pochádza.

Konštrukcia tabiel pre všeobecné konjunktívne dotazy

- Presná konštrukcia:
 - Očíslujeme predikáty v nejakom poradí
 - Očíslujeme argumenty v rámci predikátov
- Lexikograficky usporiadané argumenty tvoria hlavičku tabla (nemusíme ju explicitne vypisovať).
- Premenné hlavy dotazu tvoria sumár. Na pozíciach a poradí nezáleží.
- Pre jednotlivé podciele zostavíme riadok tabla, pričom
 - argumenty podcieľu sa musia nachádzať na správnej pozícii
 - na záver riadku pridáme tag – meno predikátu.
- Pri minimalizácii tag berieme ako konštantu, t.j. nedovoľíme subsumciu medzi riadkami s rôznymi tagmi.

Príklad: zistite vzájomné pohltenie

Q1: $p \leftarrow r(X,Y), r(X,X), r(Z,X).$

Q2: $p \leftarrow r(X,Y), r(X,X), r(X,Z).$

$r(Y, Y)$

R_{11}	R_{12}	R_{21}	R_{22}	R_{31}	R_{32}
X		X	X		X
X		X	X	X	

Optimalizácia Q1

R_{11}	R_{12}	
X		r
X	X	r
	X	r

Iné tablá pre Q1

X_1	Y	X_2	tag
X			r
X		X	r
X			r

X_1	Y	X_2	tag
X			r
X	X		r
	X		r

Pre cyklické dotazy konštrukcia alternatívnych tabiel' nie je jednoznačná.

Iný príklad

Q1: $p \leftarrow r(X, X), r(U, U).$

X X U U

Q2: $p \leftarrow r(X, Y), r(Z, X).$

X Y Z X

Q2 $\not\supseteq$ Q1 opak neplatí. Znovu sa to nedá urobiť lokálne. U sa nemôže súčasne zobrazit' na Z aj X.

Alternatívna asi lepšia konštrukcia tabla pre všeobecné CQ.

- Stĺpce konštruujeme pre premenné a pozície v predikátoch
 - Ak sa premenná vyskytuje na rôznych miestach toho istého predikátu má toľko stĺpcov koľko je takýchto výskytov.
- Riadky pre jednotlivé predikáty.

Iný príklad alternatívne

X_1	X_2	U_1	U_2	Y	Z	Tag	Query
X	X					r_1	Q1
		U	U			r_2	Q1
X				Y		r_1	Q2
	X				Z	r_2	Q2

Tie U by som tam radšej nemal resp. mal ich v stĺpcoch pre X.

Higher Normal Forms

<http://www.dcs.fmph.uniba.sk/~plachetk>

[/TEACHING/RLDB2013](http://www.dcs.fmph.uniba.sk/~plachetk/TEACHING/RLDB2013)

<http://www.dcs.fmph.uniba.sk/~sturc/databazy/rldb>

Tomáš Plachetka, Ján Šturc

Faculty of mathematics, physics and informatics,
Comenius University, Bratislava

Summer 2012–2013

Multi-valued dependency

Definition: A **multi-valued dependency (MVD)** $X \twoheadrightarrow Y$ holds in relation r , if

$$r(\mathbf{X}, \mathbf{Y1}, \mathbf{Z1}) \wedge r(\mathbf{X}, \mathbf{Y2}, \mathbf{Z2}) \Rightarrow r(\mathbf{X}, \mathbf{Y1}, \mathbf{Z2}) \wedge r(\mathbf{X}, \mathbf{Y2}, \mathbf{Z1})$$

Example:

emp(Emp_name,	Proj_name,	Boss_name)
	joe	x	fred
	joe	y	lisa
	joe	x	lisa
	joe	y	fred

$\text{Emp_name} \twoheadrightarrow \text{Proj_name}$, $\text{Emp_name} \twoheadrightarrow \text{Boss_name}$ both hold.
 $\text{Emp_name} \twoheadrightarrow \text{Emp_name}$ also holds, but it is a trivial MVD

Definition: An MVD $\mathbf{X} \twoheadrightarrow \mathbf{Y}$ is called **trivial**, when either \mathbf{Y} is subset of \mathbf{X} or $r = \mathbf{X} \cup \mathbf{Y}$

Multi-valued dependency: inference rules

IR1 (**reflexive rule for FDs**): $X \supseteq Y \Rightarrow X \rightarrow Y$

IR2 (**augmentation rule for FDs**): $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$

IR3 (**transitive rule for FDs**): $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

IR4 (**complementation rule for MVDs**):

$X \twoheadrightarrow Y \Rightarrow X \twoheadrightarrow (R - (X \cup Y))$

IR5 (**augmentation rule for MVDs**): $X \twoheadrightarrow Y \wedge W \supseteq Z \Rightarrow WX \twoheadrightarrow YZ$

IR6 (**transitive rule for MVDs**): $X \twoheadrightarrow Y \wedge Y \twoheadrightarrow Z \Rightarrow X \twoheadrightarrow (Z - Y)$

IR7 (**replication rule for FD to MVD**): $X \rightarrow Y \Rightarrow X \twoheadrightarrow Y$

IR8 (**coalescence rule for FDs and MVDs**): $X \twoheadrightarrow Y \wedge (\text{exists } W \text{ such that } W \cap Y = \emptyset \wedge W \rightarrow Z \wedge Y \supseteq Z) \Rightarrow X \rightarrow Z$

Sometimes the following notation is used (sharing the same left-hand side, similarly as with FDs): $X \twoheadrightarrow Y, Z$ stands for $X \twoheadrightarrow Y, X \twoheadrightarrow Z$. But Y and Z are now sets of attributes!

Multi-valued dependency: 4NF

Definition: A relation schema r is in **4NF** with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if for every *nontrivial* multivalued dependency $X \twoheadrightarrow Y$ in F^+ , X is a superkey in r

F^+ is a **closure** of F with respect to inference rules IR1–IR8. These inference rules are sound and complete (the soundness results from the definition of FD and MVD, the completeness is harder to prove)

Apparently, emp(Emp_name, Proj_name, Boss_name) should be decomposed into
project(Emp_name, Proj_name), boss(Emp_name, Boss_name)

Multi-valued dependency: 4NF

Definition: Relation schemas r_1 and r_2 form a **lossless (non-additive) join decomposition** of r with respect to a set F of functional *and* multivalued dependencies, if

$$(r_1 \cap r_2) \twoheadrightarrow\rightarrow (r_1 - r_2), \text{ or}$$

$$(r_1 \cap r_2) \twoheadrightarrow\rightarrow (r_2 - r_1)$$

Multi-valued dependency: 4NF

4NF decomposition algorithm

naive, but with lossless non-additive join guarantee

Input: A universal relation r and a set of functional and multivalued dependencies F

decompose(r, F)

```
{  
     $D := \{r\};$   
    while (a relation schema  $s$  exists in  $D$  which is not in 4NF)  
    {  
        find a non-trivial MVD  $\mathbf{X} \twoheadrightarrow \mathbf{Y}$  in  $s$  which violates 4NF;  
        replace  $s$  in  $D$  by relation schemas  $(s - \mathbf{Y})$  and  $(\mathbf{X} \cup \mathbf{Y})$ ;  
    }  
}
```

Join dependency

Definition. Let $\mathbf{R}, \mathbf{R}_1, \dots, \mathbf{R}_N$ denote sets of attributes. A **join dependency** is a constraint of the form $\mathbf{R} = \mathbf{R}_1 \bowtie \dots \bowtie \mathbf{R}_N$ which is satisfied if for each valid population of the relation r over attributes \mathbf{R} holds

$$r = \Pi_{\mathbf{R}_1}(r) \bowtie \dots \bowtie \Pi_{\mathbf{R}_N}(r)$$

Note that this is (indeed) a definition of lossless decomposition, i.e. global consistency. However, a **join dependency is a constraint just over sets of attributes, which does not depend on actual decomposition of database schema**

Example: $(\text{SSN}, \text{Name}, \text{ChildSSN}) = (\text{SSN}, \text{Name}) \bowtie (\text{SSN}, \text{ChildSSN})$ is implied by functional dependency $\text{SSN} \rightarrow \text{Name}$. In fact, each functional dependency implies some join dependency

We are particularly interested in **binary join dependencies** (as it turns out, these are MVDs), **which are of the form**

$$\mathbf{R} = \mathbf{R}_1 \bowtie \mathbf{R}_2$$

Again, recall that $\mathbf{R}, \mathbf{R}_1, \mathbf{R}_2$ are sets of attributes which are not necessarily actual relations of the database

Join dependency: 5NF

An MVD is a special case of a join dependency: $\mathbf{X} \twoheadrightarrow \mathbf{Y}$ in r can be equivalently expressed as $\mathbf{R} = (\mathbf{XY}) \bowtie (\mathbf{X} \cup (\mathbf{R} - \mathbf{Y}))$

Definition. A relation schema r is in **fifth normal form (5NF, or Project-Join Normal Form, PJNF)** with respect to a set F of functional, multivalued, and join dependencies if for every nontrivial join dependency $\text{JD}(R_1, R_2, \dots, R_n)$ in F^+ (i.e., implied by F) holds that every R_i is a superkey of r

Note that 5NF takes into account not only FDs and MVDs, but also all kinds of JDs

No sound and complete set of inference rules is known for JDs (some sound rules have been published, though)

Inclusion dependency

Definition: An **inclusion dependency** $r.\mathbf{A} < s.\mathbf{B}$ holds if for any valid population $\text{pop}(r)$ and any valid population $\text{pop}(s)$

$$\Pi_{\mathbf{A}}(\text{pop}(r)) \supseteq \Pi_{\mathbf{B}}(\text{pop}(s)),$$

where \mathbf{A} and \mathbf{B} are sets of attributes

Alternatively, **inclusion dependency** from relation r to relation s exists if

$$\exists \mathbf{X} \exists \mathbf{Y} (r(\mathbf{X}, \mathbf{Y}) \Rightarrow \exists \mathbf{Z} s(\mathbf{X}, \mathbf{Z}))$$

Informally: inclusion dependency states that relation r contains a set of attributes \mathbf{A} such that when \mathbf{A} attains a value in r , then the same value of \mathbf{A} must exist in relation s

Inclusion dependency

Example (a university database):

students(Student_ID, Name),

grades(Student_ID, Course_ID, Grade)

When a Student_ID appears in the relation grades, then the same Student_ID must exist in the relation students (as grades are assigned only to enrolled students)

In this example, Student_ID in grades is a foreign key for the relation students, i.e. it is a superkey in the relation students (and also vice versa). **Foreign key is a special case of an inclusion dependency.** Inclusion dependencies cover and generalise the concept of weak entity sets in ER and UML diagrams (presented in the winter semester)

Inclusion dependency

Example (an extended university database):

students(Student_ID, Name),
graduates(Student_ID, Name)
grades(Student_ID, Course_ID, Grade)

A graduated student must have received some grades, so there is an inclusion dependency from relation graduates to relation grades (but not in the opposite direction). But in this case, Student_ID is **not a foreign key** (a superkey) in the relation grades, because Student_ID \rightarrow Course_ID, Grade does not hold

Organisational (business) rules often dictate inclusion dependencies and other **integrity constraints** which must hold **for any valid population of a database**. It is desirable that the database system guards the database against violation of any known constraint. Still, many constraints either cannot be expressed in SQL or they can only be expressed in SQL in an awkward way (subsequently, their guarding increases the cost of updates)

Inclusion dependency: inference rules

IR1 (reflexivity)

$r.X < r.X$

IR2 (attribute correspondence)

if $r.X < s.Y$,

where $X = \{A_1, A_2, \dots, A_n\}$ and $Y = \{B_1, B_2, \dots, B_n\}$ and

A_i corresponds to B_i ,

then $r.A_i < s.B_i$ for $1 \leq i \leq n$

IR3 (transitivity)

if $r.X < s.Y$ and $s.Y < t.Z$, then $r.X < t.Z$

More on 4NF decomposition

Naïve approach to 4NF decomposition may produce bad database design

Example (Kifer, Bernstein, Lewis):

contracts(Buyer, Vendor, Product, Currency)

Business rule 1: If a company accepts several currencies, then each contract is described in each currency

Buyer, Vendor $\rightarrow\rightarrow$ Product, Currency

or, equivalently,

contracts = (Buyer, Vendor, Product) \bowtie (Buyer, Vendor, Currency)

Business rule 2: If two vendors supply a product, they accept some currency; and if a buyer has a contract with one of these vendors, then it has a contract with the other vendor as well

Product, Currency $\rightarrow\rightarrow$ Buyer, Vendor

More on 4NF decomposition

contracts(Buyer, Vendor, Product, Currency)

r1: Buyer, Vendor $\rightarrow\rightarrow$ Product, Currency

r2: Product, Currency $\rightarrow\rightarrow$ Buyer, Vendor

Decomposition using r1, (Buyer, Vendor, Product), (Buyer, Vendor, Currency) breaks r2, i.e. the second MVD is lost. And vice versa

Breaking an MVD by 4NF decomposition is much worse than breaking an FD by BCNF decomposition! The reason is that although e.g. (Buyer, Vendor, Product), (Buyer, Vendor, Currency) is a **lossless 4NF decomposition**, it can still contain **a lot of redundancy** if some MVD is broken:

Buyer	Vendor	Product
B1	V1	P
B2	V2	P
B1	V2	P
B2	V1	P

Buyer	Vendor	Currency
B1	V1	C
B2	V2	C
B1	V2	C
B2	V1	C

It is better not to insist on 4NF in this case

More on 4NF decomposition

Example (Kifer, Bernstein, Lewis):

dictionary(Latin, Hungarian, German, Slovak)

Business rule: If the dictionary provides translation from any language to any other (one word can have even more translations in other languages)

Latin →→ Hungarian, German, Slovak

Hungarian →→ Latin, German, Slovak

German →→ Latin, Hungarian, Slovak

Slovak →→ Latin, Hungarian, German

Again, there is no lossless 4NF decomposition which preserves all MVDs above. A workaround is to add a concept to the dictionary, i.e. to promote the dictionary to a multilingual thesaurus:

thesaurus(Concept, Descr, Latin, Hungarian, German, Slovak)

Latin → Concept, Hungarian → Concept,

German → Concept, Slovak → Concept,

Concept → Descr,

Concept →→ Latin, Hungarian, German, Slovak

(What about homonyms? Oh well...)

More on 4NF decomposition

thesaurus(Concept, Descr, Latin, Hungarian, German, Slovak)

Latin \rightarrow Concept, Hungarian \rightarrow Concept,

German \rightarrow Concept, Slovak \rightarrow Concept,

Concept \rightarrow Descr,

Concept $\rightarrow\rightarrow$ Latin, Hungarian, German, Slovak

Every FD is MVD as well. So let the decomposition begin with

Latin \rightarrow Concept and continue likewise. Resulting decomposition:

(Latin, Concept), (Latin, Hungarian), (Latin, German),

(Latin, Slovak), (Latin, Descr) is clearly 4NF. But it is too leaned towards Latin, which seems unnatural (well, perhaps not to a linguist)

More natural is to derive

Concept, Descr $\rightarrow\rightarrow$ Latin, Hungarian, German, Slovak

which yields lossless 4NF decomposition:

(Concept, Descr, Latin), (Concept, Descr, Hungarian),

(Concept, Descr, German), (Concept, Descr, Slovak)

More on 4NF decomposition

thesaurus(Concept, Descr, Latin, Hungarian, German, Slovak)

Latin \rightarrow Concept, Hungarian \rightarrow Concept,

German \rightarrow Concept, Slovak \rightarrow Concept,

Concept \rightarrow Descr,

Concept $\rightarrow\rightarrow$ Latin, Hungarian, German, Slovak

We have got:

(Concept, Descr, Latin), (Concept, Descr, Hungarian),

(Concept, Descr, German), (Concept, Descr, Slovak)

Now we can get back to FD Concept \rightarrow Descr and decompose even more:

(Concept, Descr),

(Concept, Latin), (Concept, Hungarian),

(Concept, German), (Concept, Slovak)

This is the decomposition we were after!

More on 4NF decomposition

4NF decomposition is a problem difficult to tackle formally
[Beeri and Kifer, 1986] give a universal recipe, which does not depend on a particular schema:

1. Find splitting left-hand side of MVDs
2. Find intersection anomalies
3. Apply 5-step “design strategy”

More on 4NF decomposition: 1. splitting lefthand anomaly

Definition. We say that $X \twoheadrightarrow V, W$ **splits left-hand side** of $Y \twoheadrightarrow K, L$ when both $Y \cap V$ and $Y \cap W$ are both non-empty. If there are two MVDs in the schema where one splits left-hand side of the other, then the schema exhibits **left-hand anomaly**.

Schema which exhibits a left-hand anomaly is probably wrong

For instance, in **contracts example**, the MVDs (business rules) r_1, r_2 split each other's left-hand side. It indicates that something is wrong. Indeed, r_1 and r_2 should be rather replaced by a single business rule which is the following JD:

$\text{contracts} = (\text{Buyer Product}) \bowtie (\text{Vendor Product}) \bowtie$
 $(\text{Buyer Currency}) \bowtie (\text{Vendor Currency})$

It states that buyers buy some products, vendors sell some products, buyers accept some currencies, vendor accept some currencies

More on 4NF decomposition: 2.intersection anomaly

Definition. We say that a schema suffers from **intersection anomaly** when a pair of MVDs $X \twoheadrightarrow Z$, $Y \twoheadrightarrow Z$ holds, but MVD $X \cap Y \twoheadrightarrow Z$ does not hold

If there a schema suffers from intersection anomaly, then the schema is wrong

For instance, in **dictionary example**, $\text{Latin} \twoheadrightarrow \text{Slovak}$ and $\text{Hungarian} \twoheadrightarrow \text{Slovak}$ hold, but $\emptyset \twoheadrightarrow \text{Slovak}$ does not hold. This is an intersection anomaly

A solution is usually adding a new attribute or attributes

More on 4NF decomposition: 3.design strategy

We now assume no splitting left-hand side anomalies. Then a dependency-preserving, lossless join decomposition can be produced as follows:

- a) Compute attribute closure, X^+ of every MVD $X \twoheadrightarrow Y$. Then replace the MVD $X \twoheadrightarrow Y$ with $X^+ \twoheadrightarrow Y$
- b) Find minimal cover of resulting MVDs from step a)
- c) Eliminate intersection anomalies by adding new attributes. (This can be done automatically! See [Beeri&Kifer, 1986].)
- d) Apply naïve 4NF decomposition algorithm using MVDs only
- e) Apply BCNF design algorithm using FDs only

See the *multilingual thesaurus* example

- Motivácia
- Homogénne a heterogénne databázové systémy
- Distribúované databázové systémy a transakcie
 - Požiadavky na systém, architektúra
 - Algoritmy

P.A. Bernstein, V. Hadzilacos, N. Goodman:
Concurrency Control and Recovery in Database
Systems

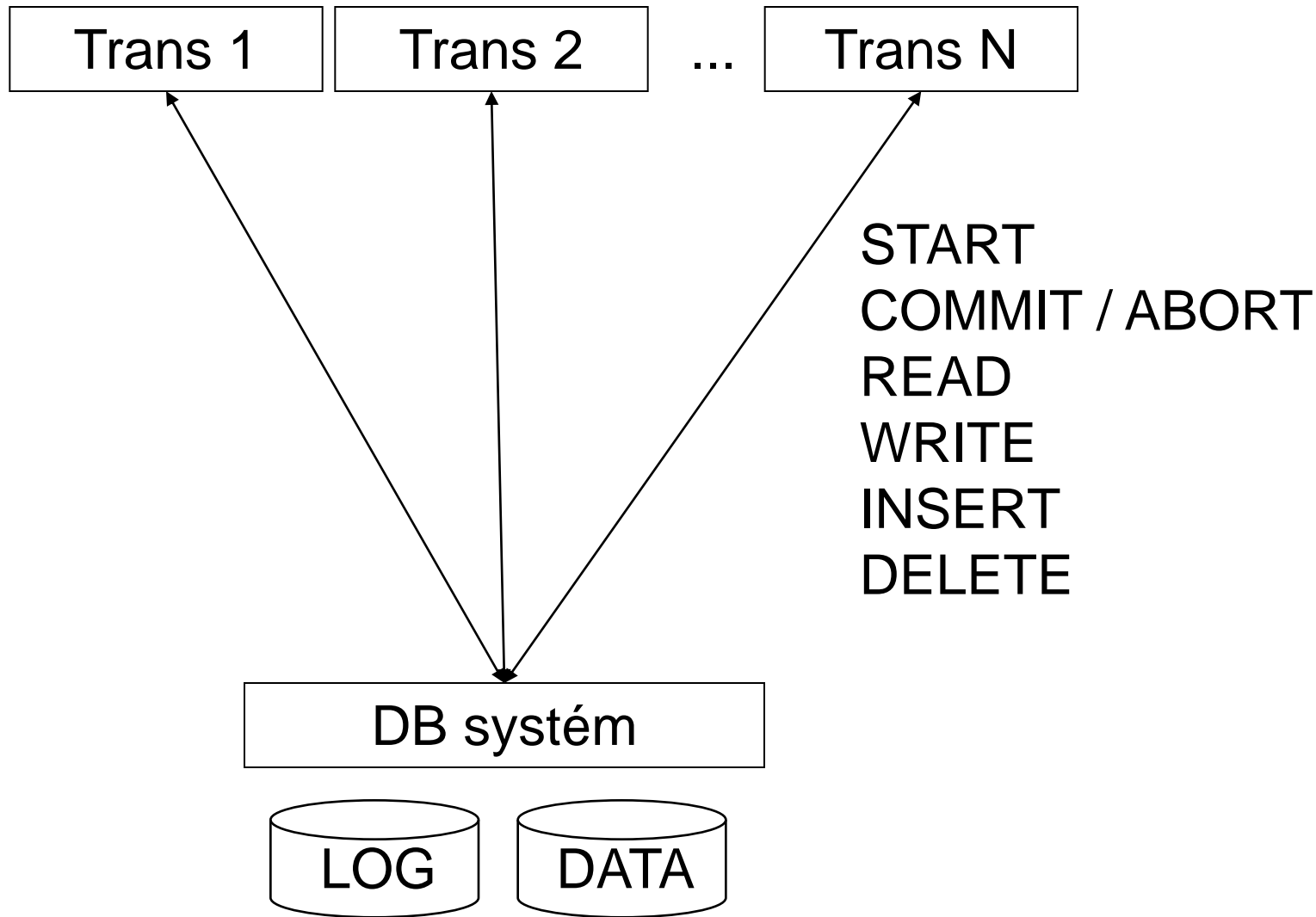
H. Garcia-Molina, J.D. Ullman, J. Widom: Database
System Implementation, Prentice Hall, 2000

Centralizovaná databáza

- Dáta sú na jednom mieste
- Centrálna správa dát a transakcií
- Architektúra klient-server (2-tier)
 - Klient posiela transakčné operácie
 - Server spracúva prichádzajúce operácie, stará sa o splnenie ACID podmienok, atď.

Takto sa to dnes robí... Asi aj preto, že iné vlastne nie je

Centralizovaná (2-tier) architektúra



Výhody centralizovanej (2-tier) architektúry:

- jednoduchosť, “**spoľahlivosť**” (v zmysle odchytenia programátorských chýb)
- kompatibilita s existujúcou byrokraciou

Nevýhody centralizovanej (2-tier) architektúry:

- všetko ostatné (**nespoľahlivosť**—ak vypadne centrála, dáta nie sú dostupné a treba počkať na obnovu; **neškálovateľnosť**—ak rýchlosť servera nestačí pre obsluhu spracovania požiadaviek, tak treba počkať na rýchlejší hardware; atď.)

- Homogénne systémy
 - Rovnaký software, rovnaká databázová schéma, rôzne dáta v rôznych uzloch
 - **Ciel': poskytnúť každému uzlu centralizovaný pohľad na dáta**
- Heterogénne systémy
 - Rôzny software, rôzne databázové schémy v rôznych uzloch
 - **Ciel': integrovať existujúce databázové systémy do funkčného celku** (napr. rezervácia letenky a zároveň hotela cez WWW u dvoch rôznych spriatelených firiem)

Je prirodzené pracovať s distribuovanými dátami:

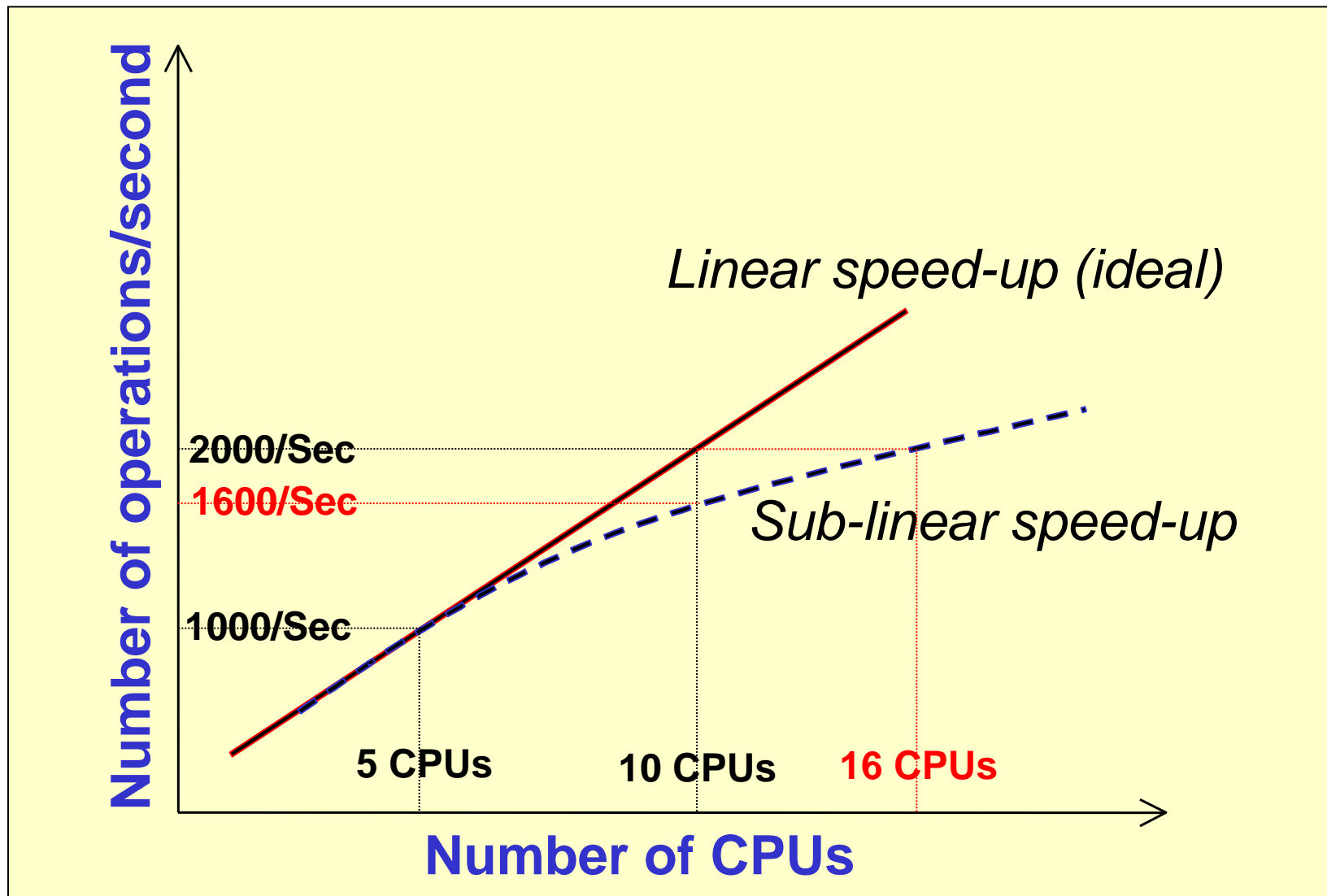
- Rôzne štátne inštitúcie pracujú s rôznymi údajmi o tých istých osobách: **vertikálna fragmentácia dát**
- Firma má veľa filiálok: **horizontálna fragmentácia dát**

Budeme sa zaoberať horizontálnou fragmentáciou.

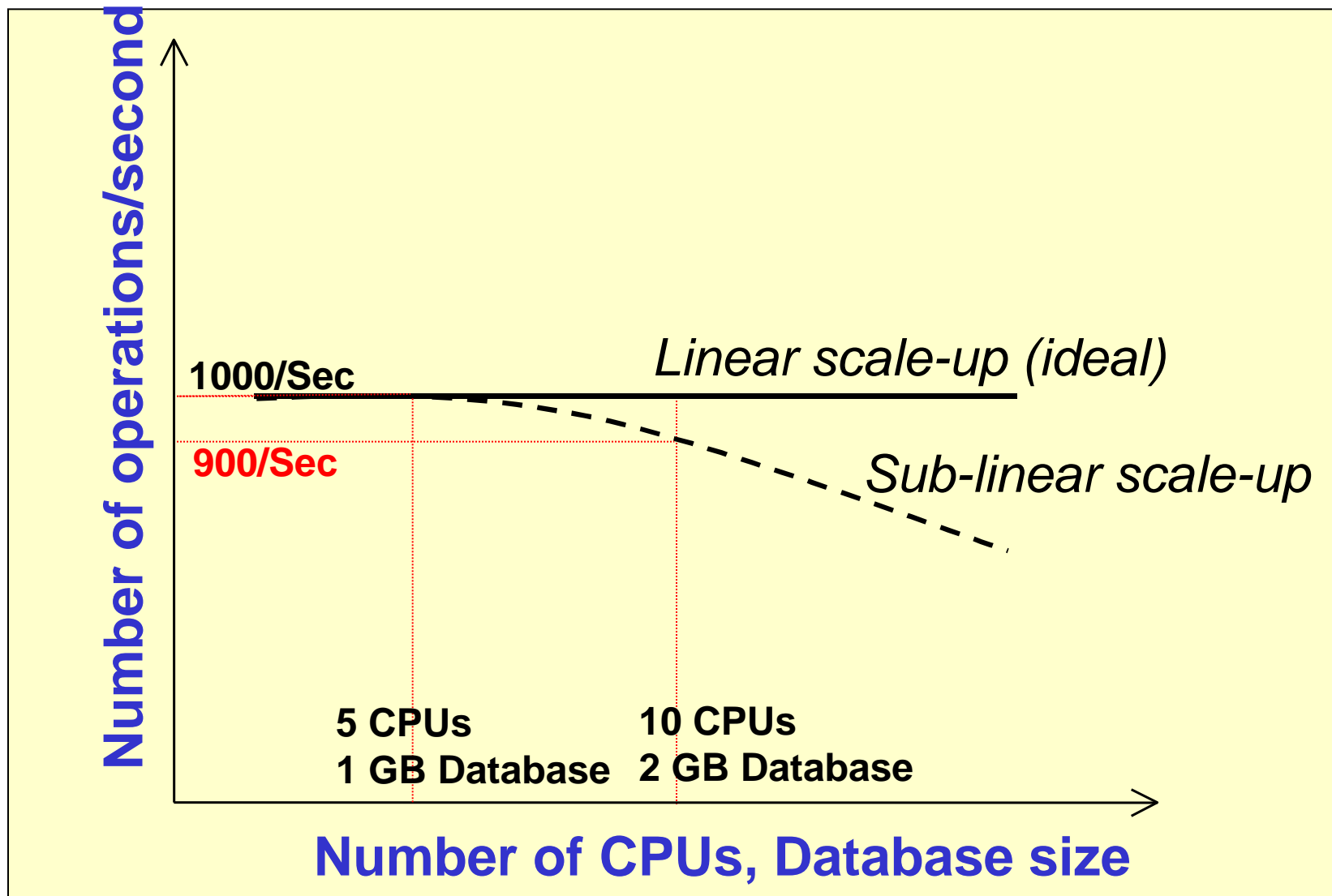
Príklad: banková pobočka má „u seba“ len účty svojich klientov. Celá banka je zjednotením filiálok

Ak klient príde do ktorejkoľvek pobočky, tak je vo svojej banke a chce vedieť narábať so svojimi účtami—tak, ako keby bol vo svojej “domácej pobočke”

SPEEDUP

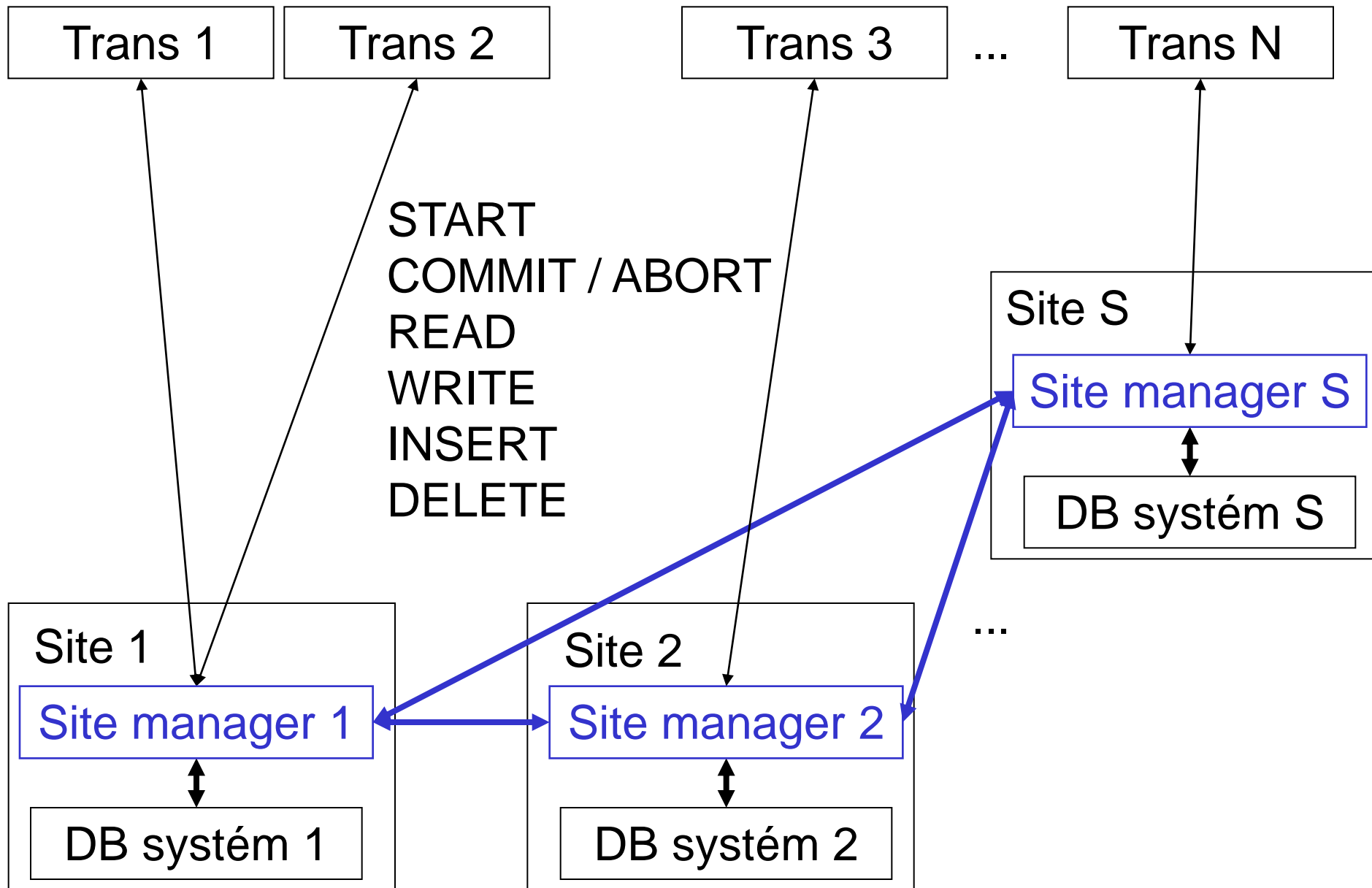


SCALEUP

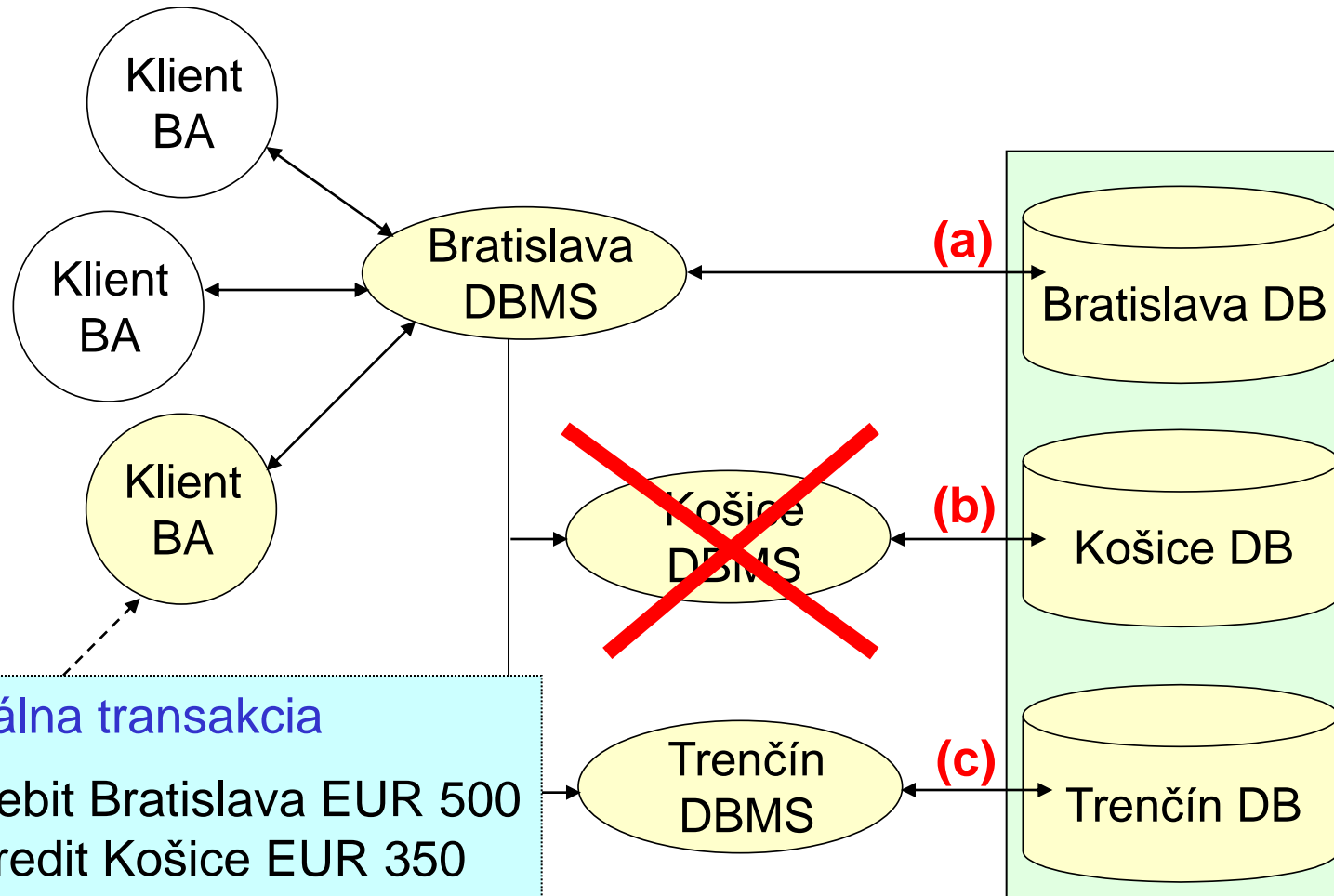


- Požiadavky na homogénny databázový systém
 - ACID
 - **Transparentnosť distribúovanosti pre aplikácie (transakcie)**
- Dodatočné požiadavky
 - Paralelizmus
 - Odolnosť voči výpadkom uzlov a komunikačných liniek: **žaden uzol nesmie nekonečne dlho čakať na obnovenie iného spadnutého uzlu**
 - Nízka réžia

Distribovaná (3-tier) architektúra



Distribúované transakcie: nové problémy



Globálna transakcia

(a) Debit Bratislava EUR 500

(b) Kredit Košice EUR 350

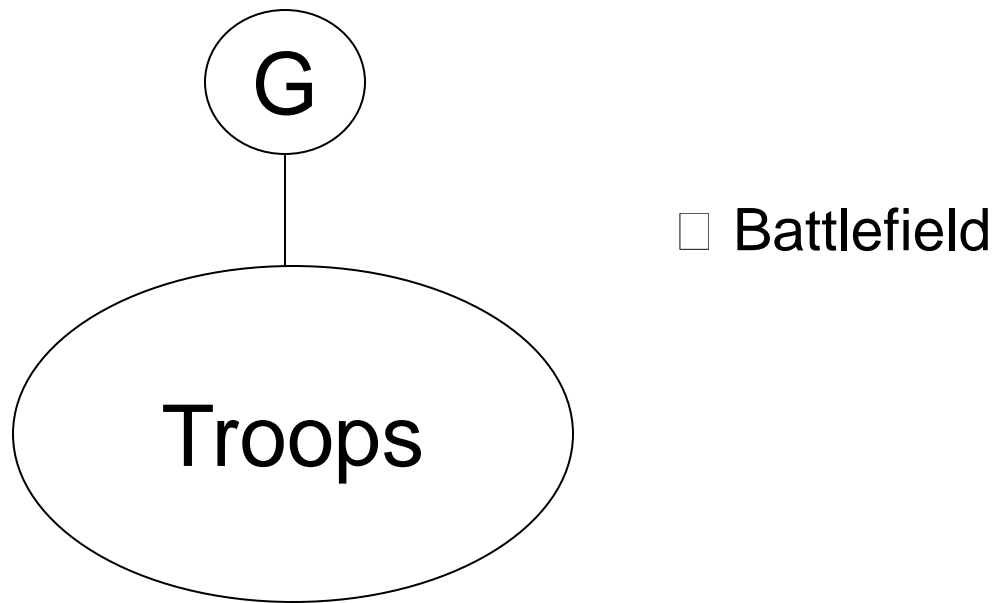
(c) Kredit Trenčín EUR 150

Nestačí splniť ACID v každom DBMS individuálne!

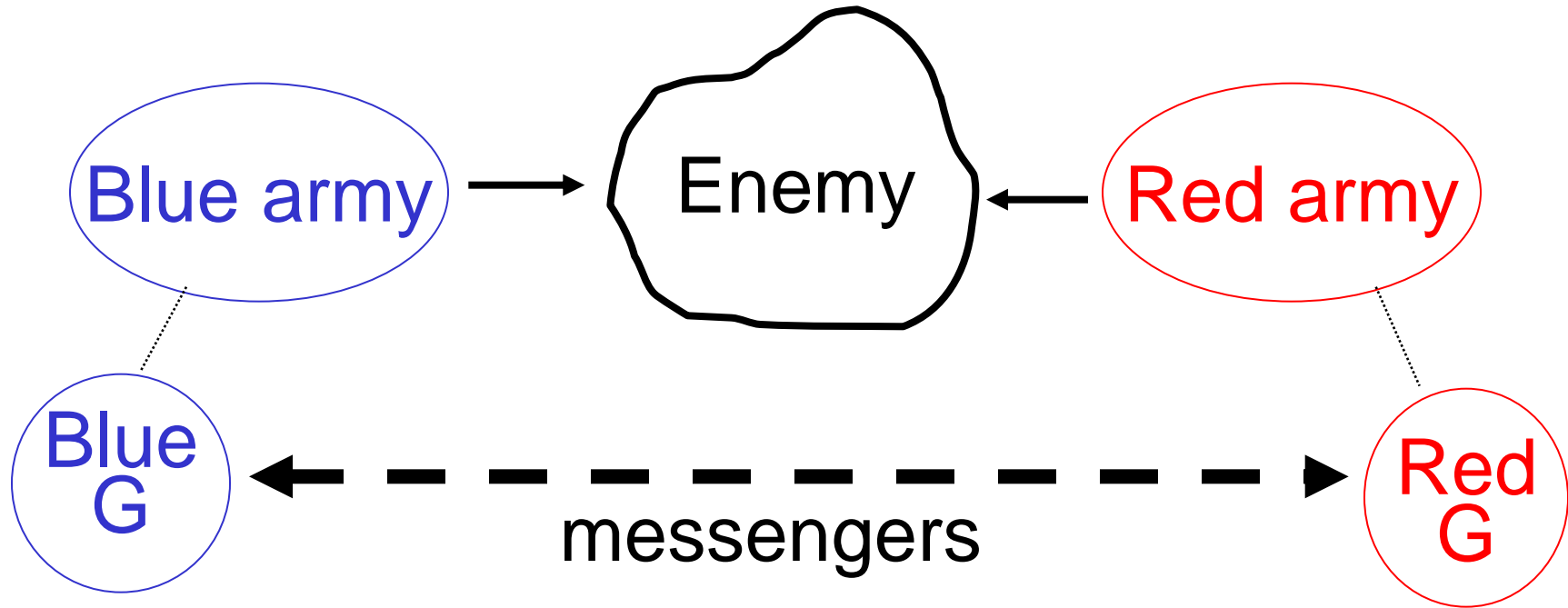
Atomický commit protokol, ciele:

- Buď sa všetky uzly dohodnú na COMMIT transakcie, alebo sa všetky uzly dohodnú na ABORT transakcie
- Predpokladáme, že ak niektorý uzol spadne alebo sa zruší niektorá z liniek, tak ten uzol jednoducho prestane komunikovať (t.j. žiadne byzantínske chyby)
- Pozor, niektorý uzol môže spadnúť **počas vykonávania commit protokolu**

The one general problem (Trivial!)



The two general problem:



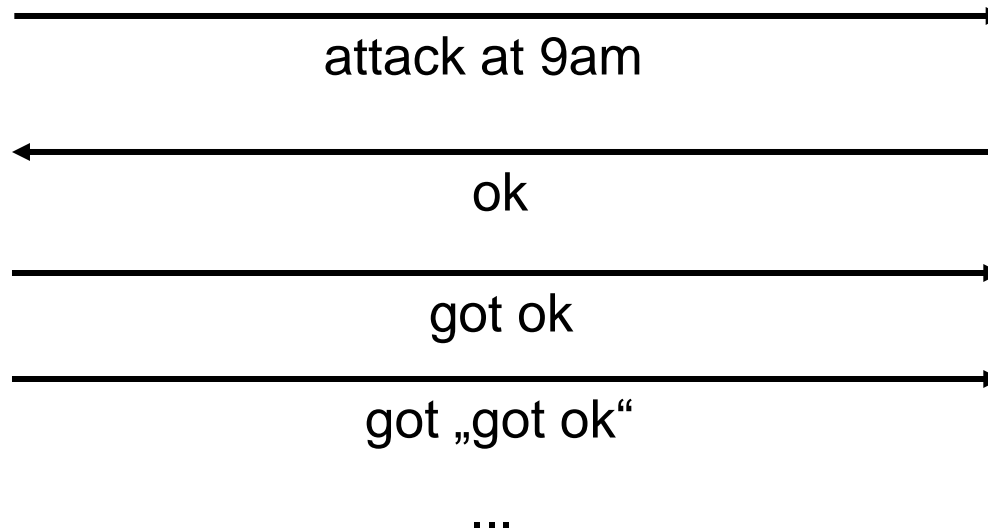
- Blue and red army must attack at same time
- Blue and red generals synchronize through messengers
- Messengers can be lost

How Many Messages Do We Need?

assume blue starts...

BG

RG



Is this enough??

There is no protocol that uses a finite number of messages that solves the two-generals problem (as stated here)

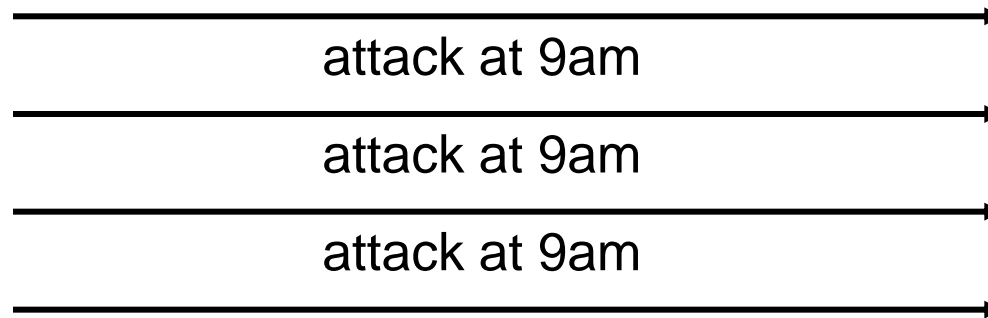
Probabilistic Approach?

- Send as many messages as possible, hope one gets through... Assume that eventually one does get through, so **keep sending until an answer arrives**

assume blue starts...

BG

RG



...

The red general does the same when he sends an answer.
When the blue general receives an answer, he knows that
his message (at least one messenger) got through!

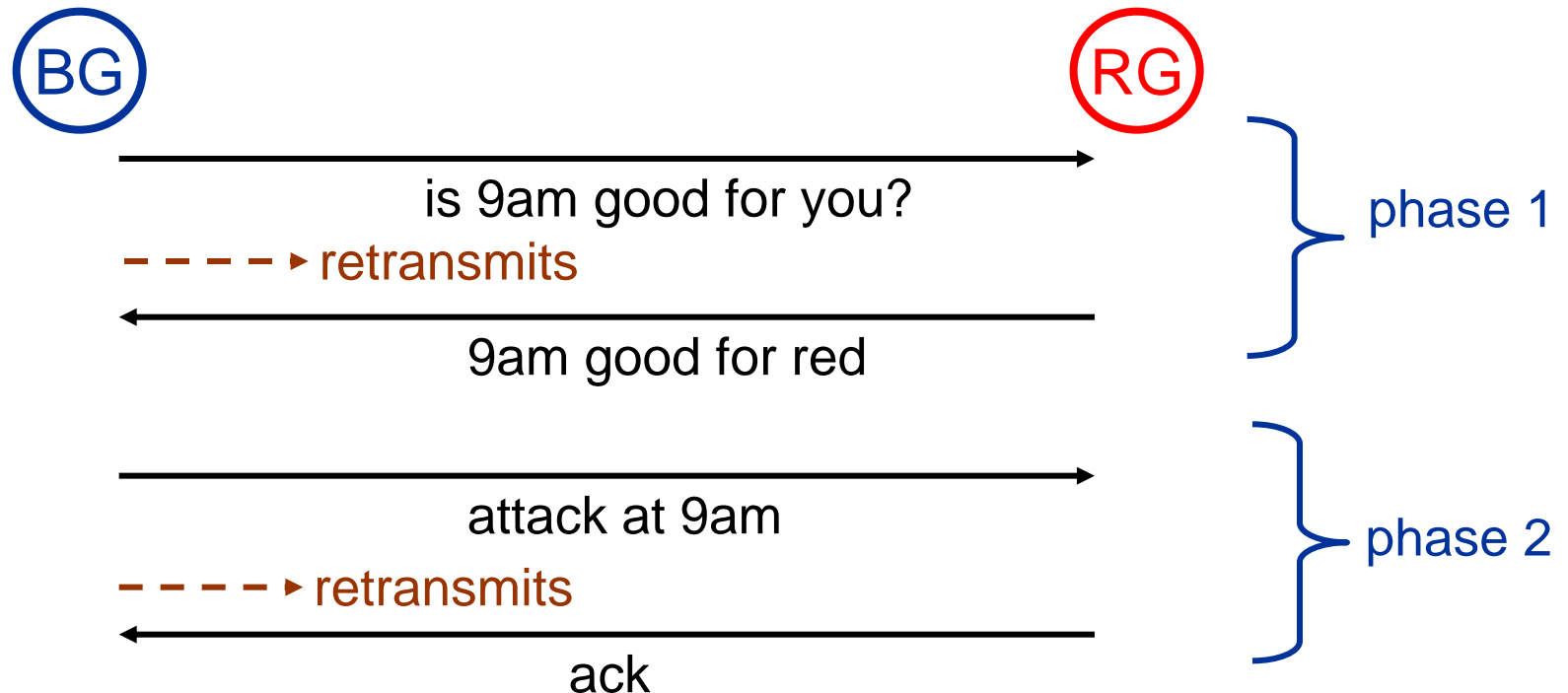
Probabilistic (?) Approach

- This is what indeed happens in contemporary networks (TCP/IP). Such a repetitive circulation of messages is hidden inside the TCP protocol. The circulation of messages (heart-beat) begins from the moment when a connection is established between two nodes, independently on whether the nodes actually attempt to communicate via that connection (ICMP packets)
- A good consequence is that from a point of view of a TCP application, the connection is reliable, i.e. messages do not get lost (and their ordering is preserved), if the connection works
- If the repeating trick stops working, then TCP gives up after a certain time has passed and breaks the connection. In such a case both the nodes are notified

Eventual Commit

- Eventually both sides attack...

assume blue starts...



This approach works for an arbitrary number of generals!

Atomický commit, formálne požiadavky (Bernstein):

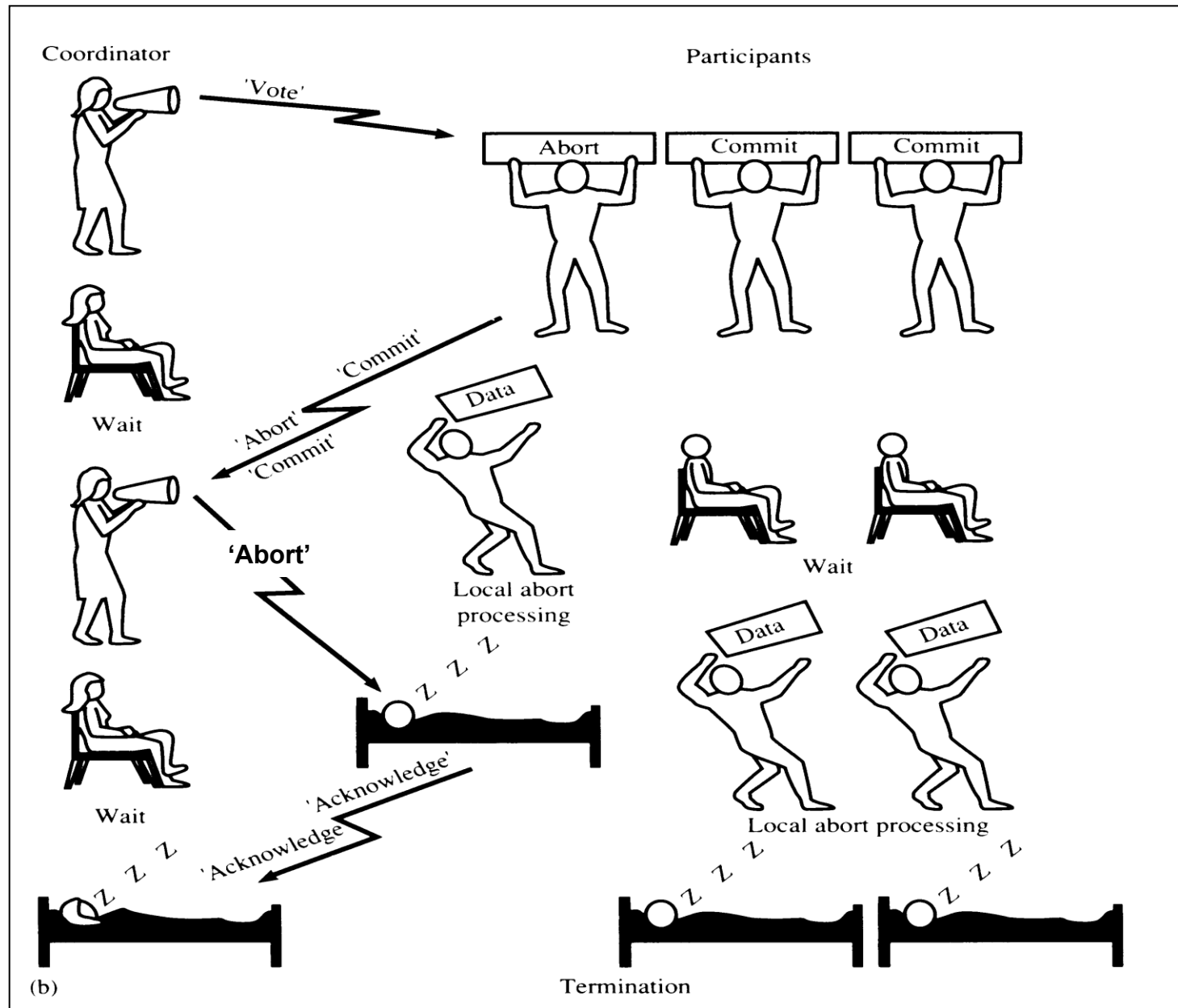
1. Všetky procesy, ktoré urobia rozhodnutie (Commit, resp. Abort), urobia rovnaké rozhodnutie
2. Keď sa proces rozhodne pre Commit, resp. Abort, tak svoje rozhodnutie už nezmení
3. Rozhodnutie Commit môže byť urobené len vtedy, ak všetky procesy hlasujú Yes (t.j. všetci s Commitom súhlasia)
4. Ak všetky procesy hlasujú Yes a nenastanú žiadne výpadky, tak rozhodnutie musí byť Commit
5. Ak po výpadku dostatočne dlho nie sú žiadne ďalšie výpadky, tak všetky procesy rozhodnutie urobia

Atomicita: atomický commit protokol

- Atomický commit protokol začína uzol, ktorý od transakcie T dostal žiadosť o commit (t.j. domovský uzol transakcie T). Tento uzol funguje ako **koordinátor**. Ostatné uzly, ktoré sa týkajú transakcie T, fungujú ako **participanti** (koordinátor vie, kto sú participanti)
- Predpokladá sa **asynchrónny model komunikácie**. To znamená, že správy sa nestrácajú; a že medzi dvojicou procesov sú doručované v poradí, v ktorom boli odoslané. Neexistuje časová garancia pre doručenie správy, ale každá odoslaná správa raz doručená bude; výnimkou je prípad, ak adresát neexistuje (je spadnutý), vtedy sa správa zahodí. Ak uzol čaká na správu od iného procesu a ten iný proces spadne (alebo vypadne spojenie medzi nimi), tak čakajúci uzol ten výpadok detekuje

Atomicity: 2-fázový atomický commit protokol

**Atomický
2-fázový
commit
protokol,
ak žiaden
uzol
nespadne:
ABORT**



2-fázový atomický commit protokol

- Ak niektorý uzol spadne a znovu sa obnoví, čo má robiť?
 - **Nutnosť loggovania intencie (zamýšľanej akcie) pred vykonaním akcie, t.j. write-ahead log. Toto je v súlade s loggovaním v centralizovaných DBMS**

2-fázový atomický commit protokol

1.fáza: transakcia T poslala koordinátorovi žiadosť o commit. Ak koordinátor sám rozhodne ABORT, tak sa táto fáza preskočí, koordinátor zapíše **<ABORT T>** do log-file a pošle správu [ABORT] všetkým participantom. Inak:

- Koordinátor zapíše do log-file <prepare T> a pošle správu [VOTE T] všetkým participantom
- Keď participant dostane správu o hlasovaní, zistí, či smie povoliť commit transakcie T:
 - Ak nie (t.j. ak participant vie lokálne rozhodnúť ABORT), zapíše **<ABORT T>** a <NO T> do log-file a pošle správu [NO T] koordinátorovi
 - Ak áno, zapíše <YES T> do log-file a pošle správu [YES T] koordinátorovi

Atomicita: 2-fázový atomický commit protokol

2-fázový atomický commit protokol

2.fáza: Koordinátor čaká na správy YES/NO od participantov

- Ak koordinátor dostane správu [NO T] od niektorého participanta, rozhodne **ABORT**, zapíše do log-file <ABORT T> a pošle participantom [ABORT T]
- Ak koordinátor dostane správu [YES T] od všetkých participantov, rozhodne **COMMIT**, zapíše <COMMIT T> do log-file a pošle participantom [COMMIT T]
- Keď participant dostane správu [COMMIT T] od koordinátora, zapíše do log-file <**COMMIT T**>, inak zapíše <**ABORT T**> (toto je rozhodnutie, ktoré sa už nikdy nezmení)

2-fázový atomický commit protokol

Finálna fáza: Koordinátor čaká na potvrdenia od participantov

- Ak koordinátor dostane správu [ACK T] od všetkých participantov, môže na transakciu T zabudnúť

Táto fáza sa zdá byť zbytočná. Lenže inak musí koordinátor naveky držať záznam o committe transakcie T (lebo jeho správu <COMMIT T>, resp. <ABORT T> nemusel niektorý participant dostať)

2-fázový atomický commit protokol

Ak spadne participant:

- Koordinátor zistí že participant spadol (detekuje timeout). V tom prípade rozhodne ABORT, zapíše <ABORT T> do log-file a pošle participantom správy [ABORT T]
- Keď sa spadnutý participant obnoví a nájde vo svojom log-file <COMMIT T>, urobí redo(T); ak nájde <ABORT T>, urobí undo(T); ak nájde <NO T>, urobí undo(T); ak nájde [YES T], tak sa informuje o výsledku hlasovania u ostatných uzlov a až na základe toho urobí undo(T) resp. redo(T).

2-fázový atomický commit protokol

Ak spadne koordinátor:

- Participant zistí že koordinátor spadol (detekuje timeout). Ak má tento participant v log-file <ABORT T> resp. <NO T>, tak sa vie rozhodnúť pre ABORT sám za seba; inak musí situáciu konzultovať s ostatnými participantmi
- Ak niektorý iný participant má v log-file <COMMIT T>, tak sa tento participant rozhodne pre COMMIT
- Ak niektorý iný participant má v log-file <ABORT T> resp. <NO T> (resp. nevie nič o hlasovaní, lebo spadol ešte skôr), tak sa tento participant rozhodne pre ABORT
- **Ak majú všetci participant v log-file <YES T>, tak sa nevedia rozhodnúť a musia čakať na obnovu koordinátora (sú zablokovaní). Toto je neakceptovateľné!**

Zaujímavý variant 2PC, decentralizovaný 2PC (len 2 kolá)

- koordinátor broadcastuje správu YES, resp. NO
- každý participant broadcastuje svoje YES, resp. NO
- po prijatí všetkých správ vie každý uzol lokálne rozhodnúť COMMIT, resp. ABORT

Atomicity: 2-fázový atomický commit protokol

Bernstein: ... “5. Ak po výpadku dostatočne dlho nie sú žiadne ďalšie výpadky, tak všetky procesy rozhodnutie urobia”

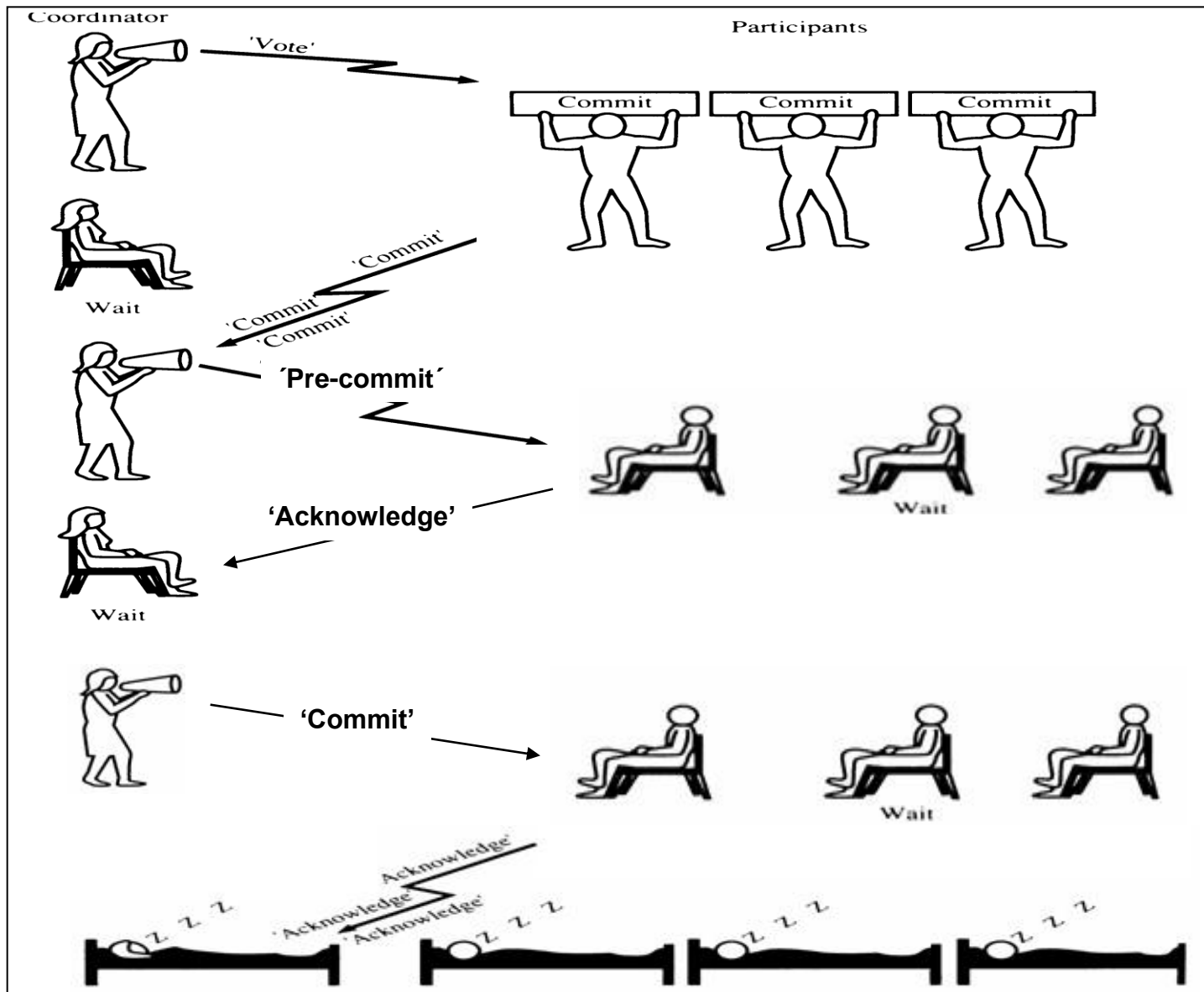
Je rozumné pridať ďalšiu požiadavku: 6. Všetky uzly, ktoré počas vykonávania atomic commit protokolu nespádnú, vedia urobiť rozhodnutie COMMIT resp. ABORT nezávisle od toho, kedy sa spadnuté uzly obnovia (t.j. aj keď sa žiaden zo spadnutých uzlov neobnoví)

2-fázový atomický commit protokol túto požiadavku nespĺňa. Ak v nevhodnom momente vypadne hoci len 1 uzol (koordinátor), tak žiaden z ostávajúcich uzlov nevie urobiť rozhodnutie skôr, ako sa koordinátor obnoví!

Neblokovací protokol existuje: stačí rozšíriť 2-fázový protokol o jednu fázu (fáza pre-commit) ⇒ 3-fázový protokol

Distribúované transakcie: atomický commit

**Atomický
3-fázový
commit
protokol,
ak žiaden
uzol
nespadne:
COMMIT**



**Atomický
3-fázový
commit
protokol,
ak žiaden
uzol
nespadne:
ABORT
(tretia fáza
je v tomto
prípade
zbytočná)**



Idea dodatočnej (prostrednej) pre-commit fázy:

- ak koordinátor nespadne, tak 3-fázový protokol prebehne ako ten 2-fázový, len má o jednu „zbytočnú“ fázu navyše
- ak koordinátor spadne, tak transakcia môže byť commitovaná len vtedy, ak niektorý z nespadnutých uzlov dostal od koordinátora správu PRE-COMMIT (toto je rozdiel oproti 2-fázovému protokolu)

Ostáva vyriešiť otázku, ako zistiť, či niekto dostal od koordinátora správu PRE-COMMIT. Odpoveď: nespadnuté uzly si zvolia nového koordinátora. Ak má PRE-COMMIT nový koordinátor, nemusí sa nič pýtať; ak nie, opýta sa všetkých nespadnutých uzlov, stačí keď nájde jeden, ktorý PRE-COMMIT dostal

Ak nikto PRE-COMMIT nemá, nový koordinátor rozhodne ABORT; ak má, tak **najskôr** rozšíri PRE-COMMIT medzi všetky nespadnuté uzly a až potom rozhodne COMMIT

Ani 3-fázový protokol nerieši všetky problémy:

- Ak zlyhanie liniek rozdelí uzly do dvoch izolovaných podgrafov, tak izolované komponenty môžu urobiť rôzne rozhodnutia! Hrozí inkonzistencia!

Riešenie: **Majoritný 3-fázový protokol**

- treba pridať správu pre-abort [Bernstein ~1987]
- transakcia môže byť commitovaná, ak väčšina uzlov má pre-commit; môže byť abortovaná, ak väčšina uzlov má pre-abort; inak treba počkať na obnovu spadnutých uzlov (blokovanie)
- technický problém: po obnove treba niektoré uzly “konvertovať” zo stavu commitable na stav abortable alebo naopak; dá sa to
- garantuje konzistenciu v prípade akýchkoľvek výpadkov
- za istých okolností sa zablokuje, ale menej často ako 2-fázový protokol [Skeen ~1982]

Oracle (a aj iné systémy) implementuje 2-fázový protokol, lebo je jednoduchší na implementáciu; a pritom sa spolieha na prozreteľnosť, ktorá snád' nenechá koordinátora zlyhať v nesprávnom momente príliš často

Distribúované databázy (pokračovanie)

- Výber koordinátora
- Replikácia dát
 - Distribúvaný locking
 - Distribúvaná správa deadlockov
- Distribúvané časové pečiatky (time-stamps)
- Synchronizácia času

Literatúra:

P.A. Bernstein, V. Hadzilacos, N. Goodman: Concurrency Control and Recovery in Database Systems,
<http://research.microsoft.com/pubs/ccontrol/>

H. Garcia-Molina, J.D. Ullman, J. Widom: Database System Implementation, Prentice Hall, 2000

A.S. Tanenbaum: Distributed Operating Systems, Prentice Hall, 1995

Výber koordinátora

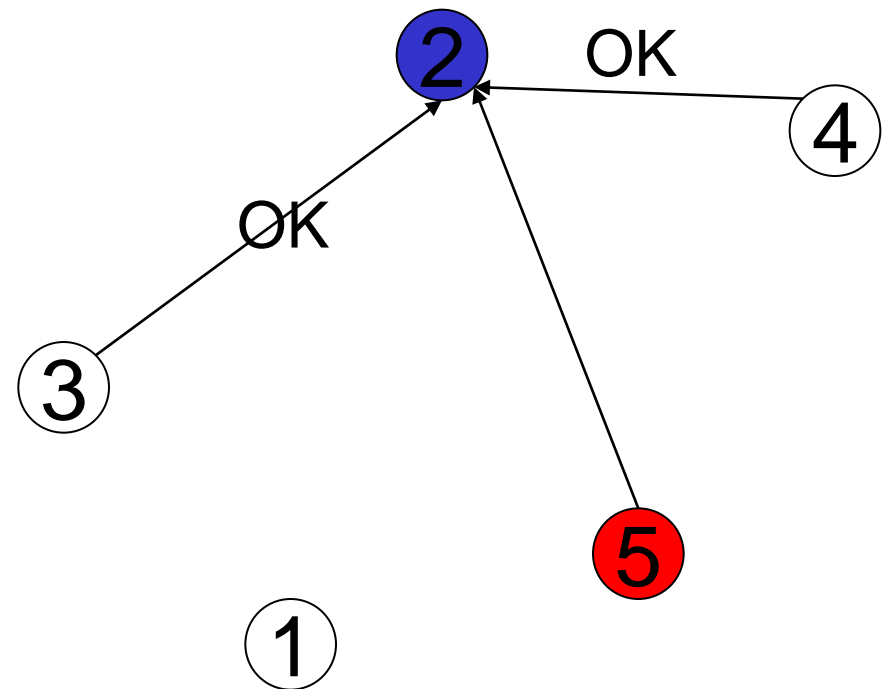
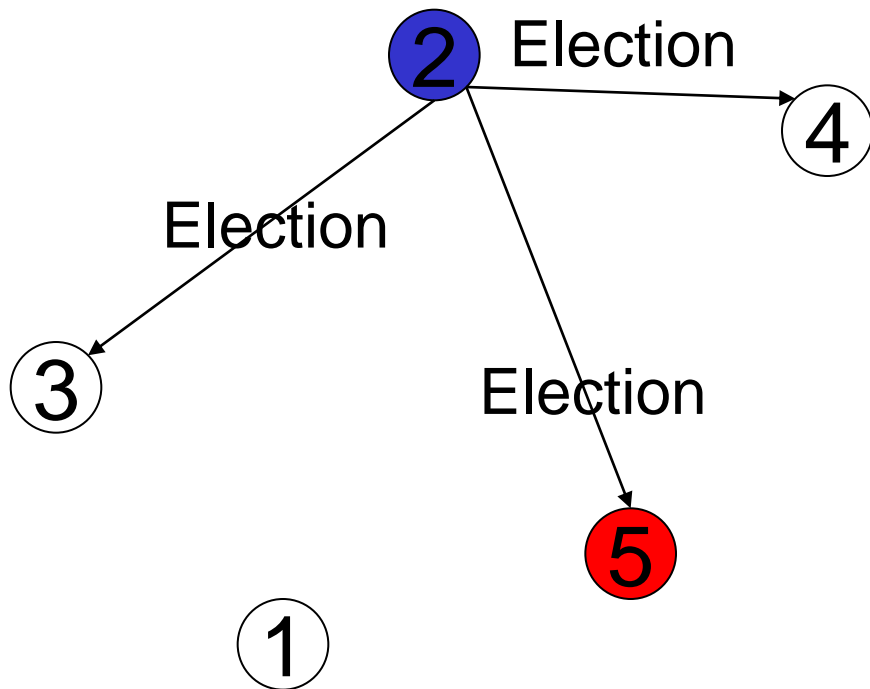
Pravidlá hry:

- Každý uzol má jednoznačný identifikátor
- Každý uzol pozná všetky identifikátory
- Komunikačný graf je v prípade výpadkov uzlov či liniek súvislý

Cieľ: Koordinátorom bude **nespadnutý** uzol s najväčším identifikátorom

Bully algoritmus

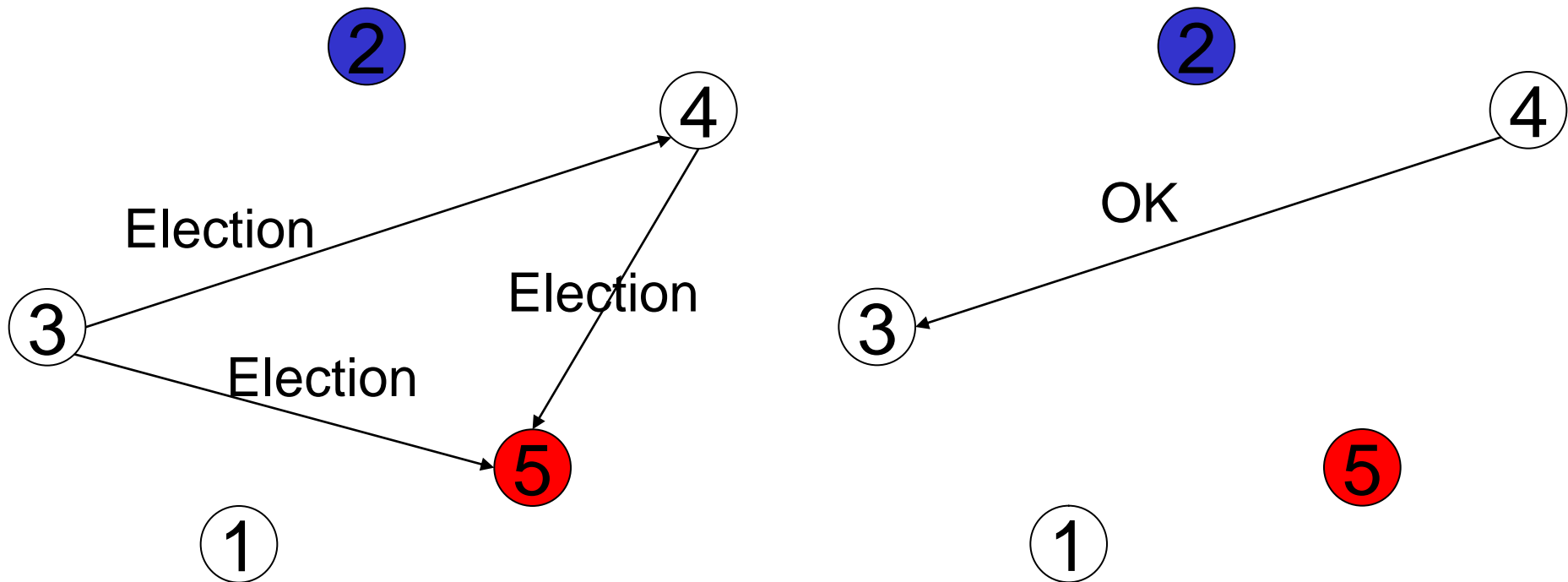
- Príklad: uzol 2 je iniciátor, uzol 5 je spadnutý



Výber koordinátora

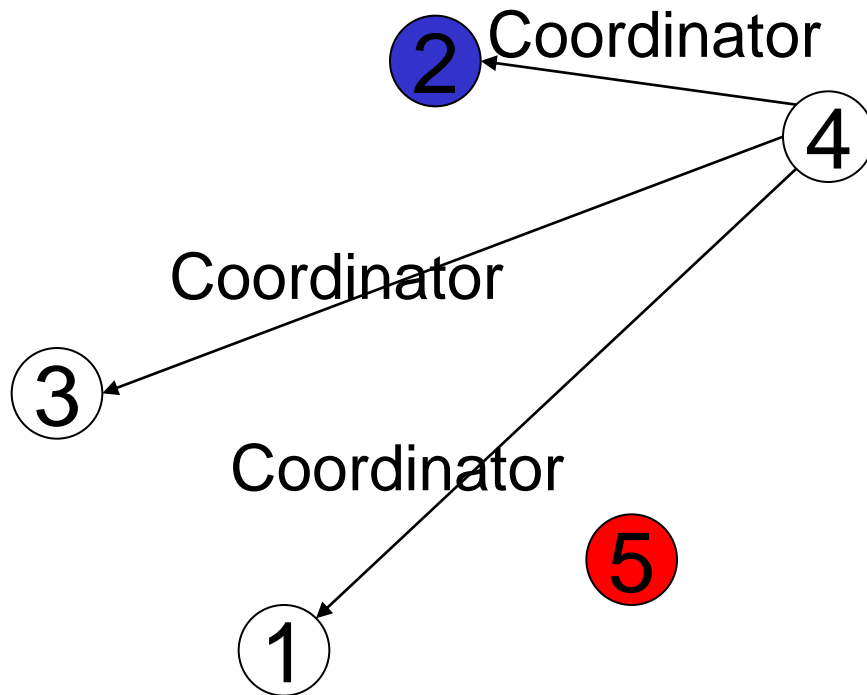
Bully algoritmus

- Príklad: uzol 2 je iniciátor, uzol 5 je spadnutý



Bully algoritmus

- Príklad: uzol 2 je iniciátor, uzol 5 je spadnutý



Pravidlá hry:

- Rovnaké dáta sú replikované vo viacerých uzloch
- Čítanie dát (read) je možné z ľubovoľného z týchto uzlov, ale prepisovanie dát (write) treba urobiť vo všetkých kópiách

Cieľ: Zabezpečiť sériovateľnosť (izoláciu) transakcií

→ distribuované zamykanie, distribuované časové pečiatky

Centralizovaná schéma

- Jeden uzol je lock manager pre všetky transakcie
 - Jednoduché na implementáciu
 - Malá odolnosť voči chybám, bottleneck

Distribúované zamykanie (2-fázový locking)

Primary-copy schéma

- Každý uzol je lock manager, zodpovedný za podmnožinu dát
 - V prípade replikovaných dát, **práve jedna z kópií je primárna (t.j. jeden uzol spravuje primárnu kópiu)**
 - O read-lock na dáta je možné žiadať ľubovoľného lock managera, ktorý má kópiu tých dát
 - O write-lock treba žiadať primárneho lock managera, no ten musí žiadosť konzultovať s ostatnými lock managermi, ktorí majú kópiu dát
 - Menší bottleneck ako u centralizovaného algoritmu
 - Možnosť distribuovaného deadlocku
- Primary-copy schéma je zhruba rovnako dobrá resp. zlá ako plne distribuovaná schéma

Distribúované zamykanie (2-fázový locking)

2 extrémálne protokoly (plne distrib. schéma):

Distribúvaný 2-fázový ROWA protokol, Read-One-Write-All

- O read-lock na dáta je možné žiadať ľubovoľného lock managera, ktorý má kópiu tých dát
- O write-lock treba žiadať všetkých ostatných lock managerov, ktorí majú kópiu dát

Majoritný 2-fázový protokol

- O read- či write-lock na dáta treba získať súhlas **väčšiny** lock managerov, ktorí majú kópiu tých dát
- $2(n/2 + 1)$ správ pre lock, $(n/2 + 1)$ správ pre unlock
- Deadlock môže nastať aj pri zamykaní 1 záznamu: napr. každá z 3 transakcií môže vlastniť zámky na 1/3 kópií toho záznamu (tomuto sa dá vyhnúť, ak je dopredu dané poradie uzlov, ktorým sú posielané žiadosti o zámok)

Distribúované zamykanie (2-fázový locking)

Quorum protokol: Kompromis medzi ROWA a majoritným protokolom (plne distrib. schéma):

- Každému uzlu je priradená nejaká váha w_i , $w_i > 0$
- Označme S sumu všetkých váh: $S = \sum w_i$
- Nech Q_r (read quorum) a Q_w (write quorum) sú nezáporné čísla také, že $Q_r + Q_w > S$ a zároveň $2 Q_w > S$
- Q_r a Q_w môžu byť dokonca rôzne pre rôzne záznamy
- Každý read musí získať zámok na toľkých kópiách, aby suma uzlov, ktoré tie kópie spravujú, bola aspoň Q_r
- Každý write musí získať zámok na toľkých kópiách, aby suma uzlov, ktoré tie kópie spravujú, bola aspoň Q_w

Distribúované zamykanie (2-fázový locking)

Quorum protokol: intuícia

- $Q_r + Q_w > S$

garantuje, že každé read a write quorum sa prekrývajú v aspoň jednom uzle (ktorý má aktuálnu hodnotu čítaného, resp. zapisovaného objektu). Navyše, nedá sa súčasne vytvoriť read a write quorum pre rovnaký objekt

- $2 Q_w > S$

garantuje, že je nemožné vytvoriť súčasne dve rôzne write quora pre rovnaký objekt

Pripomína to pravidlá pre zamykanie, len na vyššej úrovni

Distribovaná správa deadlockov

- Správa deadlockov je v distribuovanom systéme komplikovanejšia, lebo deadlocky môžu byť distribuované

Príklad: T1 beží na uzle 1, T2 beží na uzle 2

Uzol 1 (spravuje X)

write-lock1(X)

w1(X)

write-lock1(Y)

Uzol 2 (spravuje Y)

write-lock2(Y)

w2(Y)

write-lock2(X)

- T1 a T2 sú v deadlocku
- Lenže uzol 1 vie len to, že T2 čaká na T1, a uzol 2 vie len to, že T1 čaká na T2!

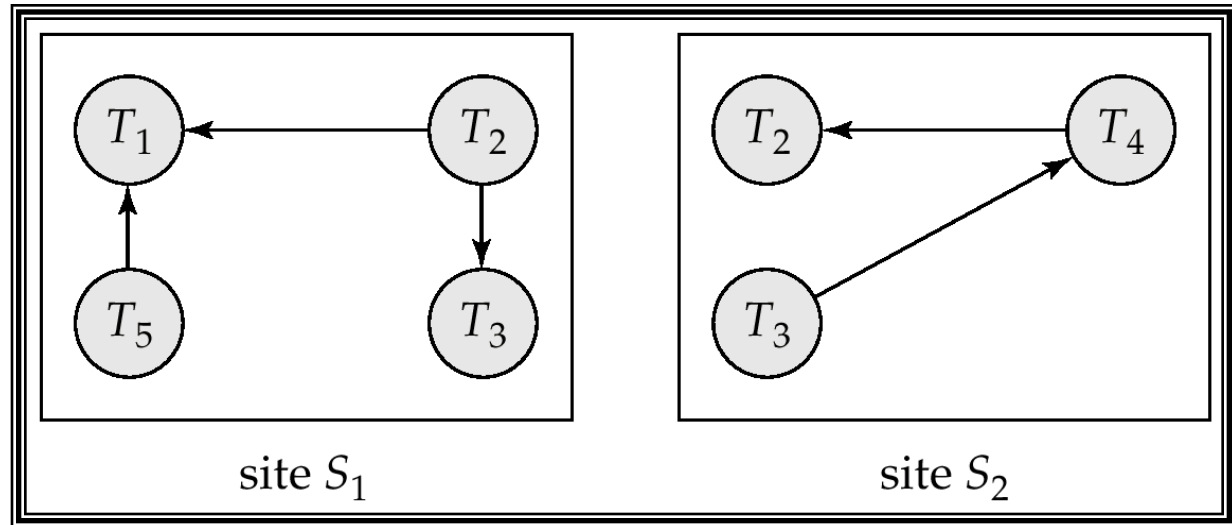
Distribovaná správa deadlockov

- Riešenie: detekcia distribuovaných deadlockov a následný abort transakcií, ktoré spôsobujú deadlock
- Správa **lokálneho wait-for-grafu (WFG)** v každom uzle
- Distribuovaný protokol na konštrukciu **globálneho WFG**, ktorý kombinuje lokálne WFG
- Deadlock sa prejaví ako **cyklus v globálnom WFG**
- Globálny WFG konštruuje jeden uzol: **koordinátor**

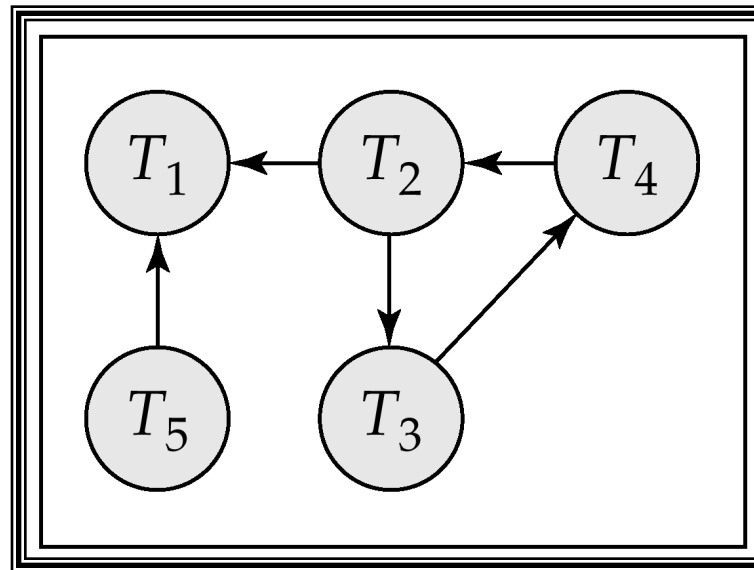
Distribovaná správa deadlockov

Príklad

Lokálne WFG



Globálny WFG



Protokol pre konštrukciu globálneho WFG

- Koordinátor pošle žiadosť o lokálne WFG
- Koordinátor skombinuje lokálne WFG do globálneho WFG a abortuje transakcie tak, aby odstránil cykly v globálnom WFG
- Abortovanie transakcií musí koordinátor oznámiť všetkým ostatným uzlom (treba skombinovať s atomickým commit protokolom)

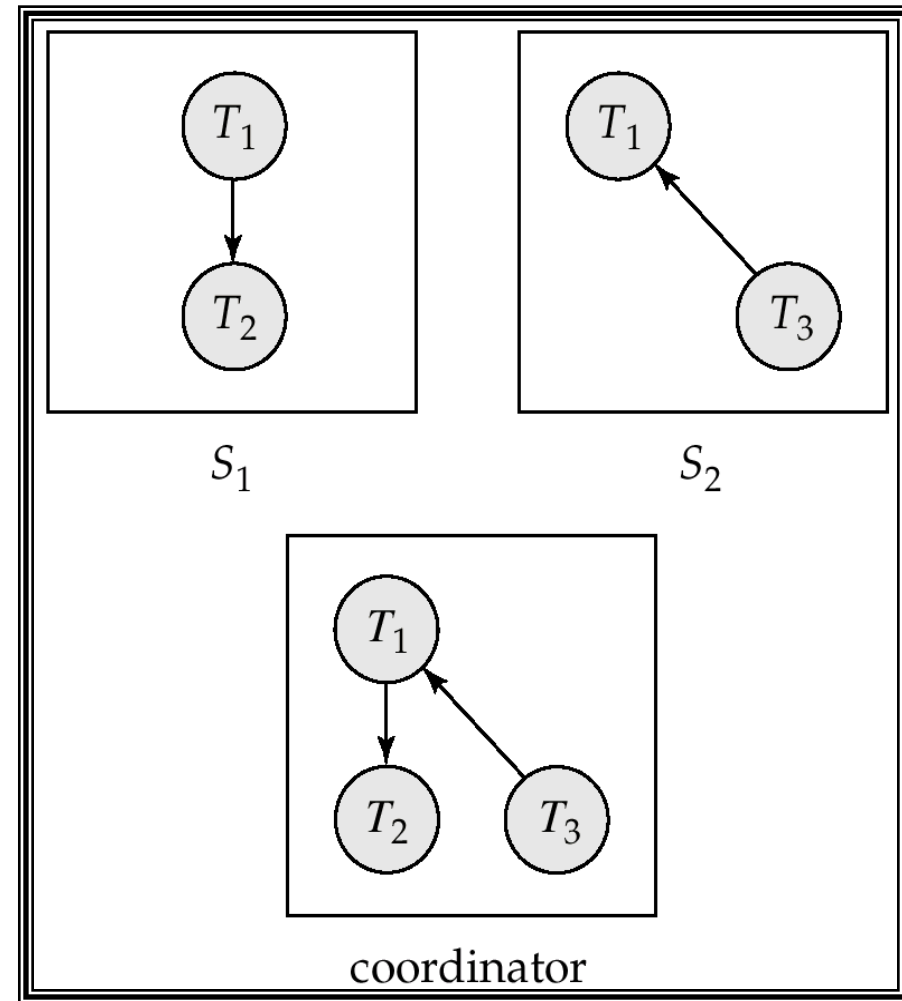
Problém:

- Koordinátor nepozná *skutočný* globálny WFG, len jeho *aproximáciu* (kvôli komunikačným oneskoreniam)
- Dôsledkom je detekcia cyklov, ktoré v skutočnosti neexistujú
- To znamená, že koordinátor niekedy abortuje transakciu, ktorú abortovať nemusí

Distribovaná správa deadlockov

Príklad: Detekcia neexistujúcich cyklov

- T2 uvoľní zámok v uzle S1: hrana $T1 \rightarrow T2$ má byť zrušená
- T2 požiada o zámok, ktorý vlastní T3 v uzle S2: v uzle 2 pribudne hrana $T2 \rightarrow T3$
- Lenže koordinátor môže tieto 2 udalosti vidieť v opačnom poradí, ak mu S2 pošle svoj lokálny WFG skôr ako S1, vtedy detekuje cyklus $T1 \rightarrow T2 \rightarrow T3 \rightarrow T1$ v globálnom WFG a zbytočne abortuje niektorú z T1, T2, T3



Synchronizácia času (fyzické hodiny)

Pravidlá hry

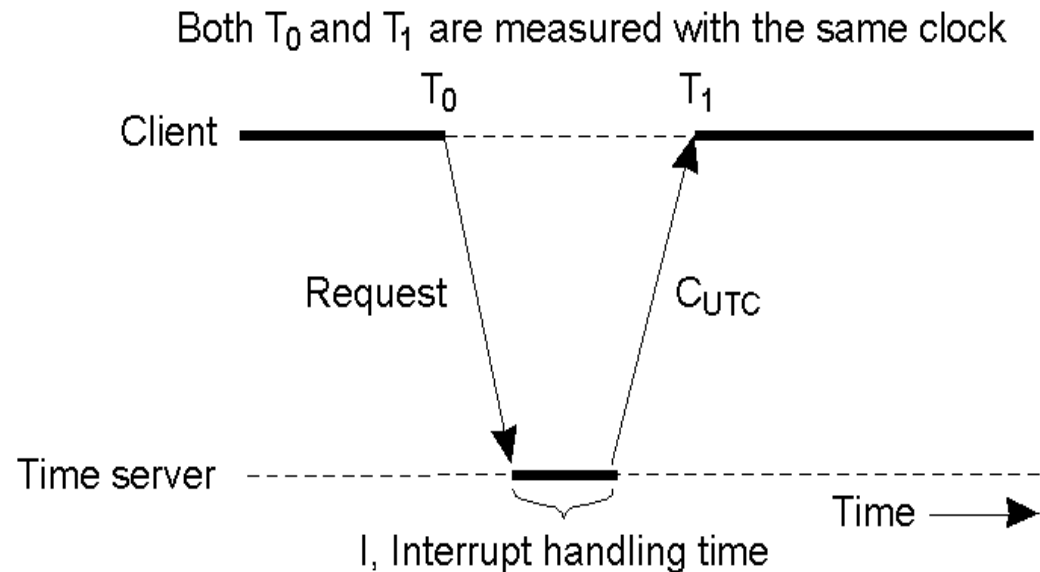
- Každý uzol má svoje lokálne hodiny
- Všetky hodiny idú vždy len dopredu, nikdy dozadu!
- Rôzne hodiny môžu ukazovať rôzne fyzické časy
- Rôzne hodiny môžu tikať rôzne rýchlo

Úlohou je hodiny všetkých uzlov synchronizovať, t.j. nastaviť (čo najpresnejšie) rovnaký čas vo všetkých uzloch

Synchronizácia času (fyzické hodiny)

Christianov protokol

- Jeden z uzlov je time-server, podľa ktorého sa všetky ostatné uzly synchronizujú

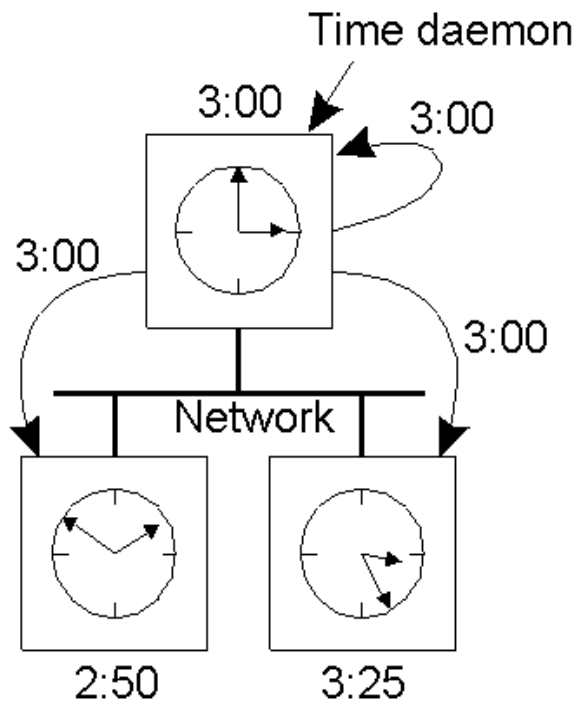


- Odhad komunikačného času: $(T_1 - T_0) / 2$, ak je I neznáme
- Odhad komunikačného času: $(T_1 - T_0 - I) / 2$ ak je I známe
- Ak je čas time-servera C_{UTC} väčší ako čas klienta T_1 , tak klient posunie svoje hodiny dopredu na $C_{UTC} + (T_1 - T_0 - I) / 2$, inak klient na istý čas Δt **spomalí** svoje hodiny (hodiny musia ísť vždy vpred)

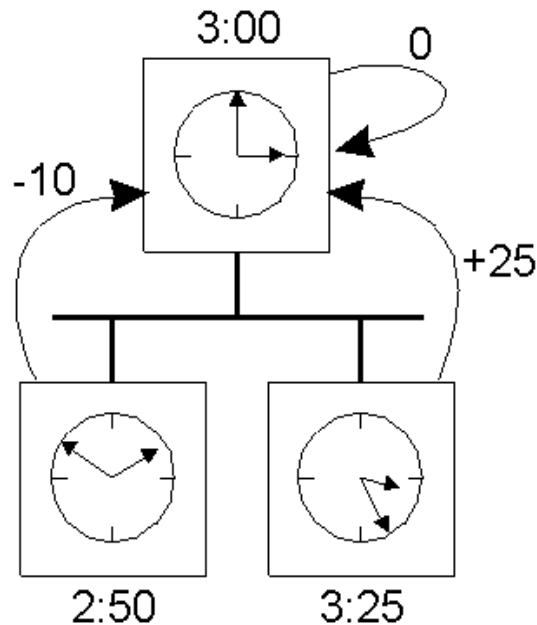
Synchronizácia času (fyzické hodiny)

Berkeley protokol

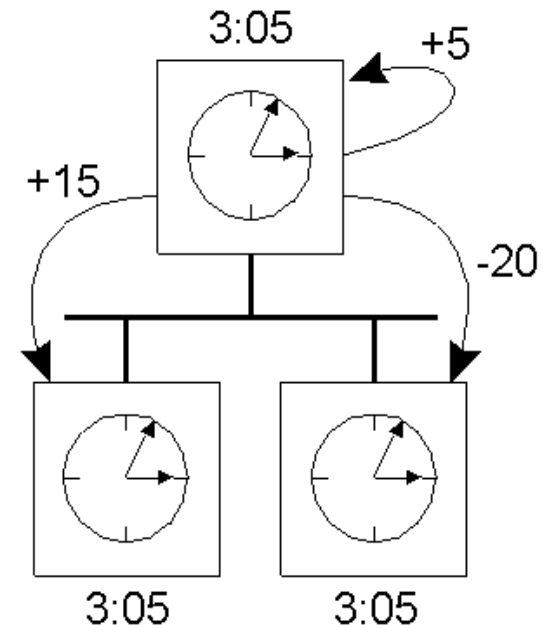
- Jeden z uzlov je time-server, ktorý sa synchronizuje podľa klientskych hodín (opak Christianovho prístupu)
- Time-server priemeruje svoje hodiny s hodinami klientov a koriguje všetky fyzické časy vrátane svojho



(a)



(b)



(c)

Synchronizácia času (logické hodiny)

Pravidlá hry

- Nie je dôležité aký čas ktoré hodiny ukazujú, dôležité je len poradie udalostí (napr. správa nebola prijatá skôr ako bola odoslaná)
- Relácia $A \rightarrow B$ znamená “udalosť A sa stala pred udalosťou B”
- $C(A)$: logický (nejaký) čas udalosti A

Úlohou je priradiť časové pečiatky $C(A_i)$ udalostiam A_i ($i=1, \dots, \infty$) tak, aby platili nasledujúce požiadavky:

- Ak udalosť A_i nastala pred udalosťou A_j v jednom procese, tak potom $C(A_i) < C(A_j)$
- Ak A_i znamená prijatie správy v nejakom procese a A_j znamená následné odoslanie správy z toho istého procesu, tak potom $C(A_i) < C(A_j)$
- Pre všetky pridelené časové pečiatky, ktoré popisujú sled udalostí, platí $C(A_i) \neq C(A_j)$

Synchronizácia času (logické hodiny)

Lamportov protokol, príklad implementácie (korekcia lokálnych hodín podľa prichádzajúcej časovej pečiatky)

Fyzické hodiny v 3 procesoch

0	0	0
6	8	10
12	16	20
18	24	30
24	32	40
30	40	50
36	48	60
42	56	70
48	64	80
54	72	90
60	80	100

Časové pečiatky (nekorektné):
6, 16, 24, 40, 60, **56**, 64, **54**

Logické hodiny v 3 procesoch

0	0	0
6	8	10
12	16	20
18	24	30
24	32	40
30	40	50
36	48	60
42	61	70
48	69	80
70	77	90
76	85	100

Časové pečiatky (korektné):
6, 16, 24, 40, 60, 61, 69, 70

Synchronizácia času (logické hodiny)

Lamportov protokol, alternatívna implementácia (bez korekcie lokálnych hodín)

- Udalosti A, ktorá nastala v uzle S a každej správe odosielanej z uzla S sa priradí vektorová časová pečiatka

$C(A)=[c_1, c_2, \dots, c_S(\mathbf{A}), \dots, c_N]$, kde

- $c_S(A)$ je fyzický lokálny čas udalosti A v uzle S
- c_i je najvyšší fyzický čas uzla i ($i \neq S$), ktorý je známy uzlu S, t.j. najvyšší čas na i-tej pozícii vektorových časových pečiatok všetkých správ, ktoré uzol S prijal pred udalosťou A
- N je počet uzlov

- Sled udalostí je popísaný lexikografickým usporiadaním vektorových časových pečiatok
- Synchronizácia fyzických hodín nie je nutná (nutné je len to, aby fyzické hodiny všetkých uzlov išli vždy len dopredu)